Knights and Knaves

A special island is inhabited by knights and knaves.

A knight always tells the truth

A knave always lies

How to Solve a Knights and Knaves problem

Example Problem:

You meet two people on the island

Kenta and Reethi

Kenta tells you: ① "Reethi is a knave"

Reethi says: ② "Neither Kenta nor I are knaves"

Step 1: Simplify the statements

Remove any negations (not, neither) if you can

In this example, statement ① is already simplified

But statement ② can be simplified,

Reethi says "Neither Kenta nor I are knaves"

This is the same as:

Reethi says "Kenta and I are both knights"

Step 2: Write the corresponding statements that would be true if each person is a knave

① In this example Kenta says "Reethi is a knave"

If Kenta is a knight:

then the statement "Reethi is a knave" is true
and Reethi must be a knave

If Kenta is a knave

then the statement "Reethi is a knave" is a lie
and Reethi must not be a knave
so Reethi must be a knight.

② In this example, Reethi says "Neither Kenta nor I are knaves"

The simplified version is "Kenta and I are both knights"

If Reethi is a knight:

then the statement "Kenta and I are both knights" is true
and Reethi and Kenta are both knights.

If Reethi is a knave:

then the statement "Kenta and I are both knights" is a lie
and either:
Kenta is a knight and Reethi is a knave
OR Kenta is a knave and Reethi is a knight
OR Kenta and Reethi are both knaves

Step 3: Determine how many cases there are

This can be done without looking at the statements

In this example, there are 2 people (Kenta and Reethi)

The cases are:

A) Both Kenta and Reethi are knights
B) Both Kenta and Reethi are knaves
C) Kenta is a knight, Reethi is a knave
D) Kenta is a knave, Reethi is a knight

There are four cases. You can think of the cases as the "possible worlds" we could exist in.

If there were 3 people: Zoey, Paola, and Minsuk

The cases would be:

A) Everyone is a knight
B) Everyone is a knave
C) Zoey and Paola are knights, Minsuk is a knave
D) Zoey and Minsuk are knights, Paola is a knave
E) Paola and Minsuk are knights, Zoey is a knave
F) Zoey is a knight, Paola and Minsuk are knaves
G) Paola is a knight, Zoey and Minsuk are knaves
H) Minsuk is a knight, Zoey and Paola are knaves

In this version, there are eight cases.

It might be helpful to visualize these cases with a table
(next page)

A) Everyone is a knight
B) Everyone is a knave
C) Zoey and Paola are knights, Minsuk is a knave
D) Zoey and Minsuk are knights, Paola is a knave
E) Paola and Minsuk are knights, Zoey is a knave
F) Zoey is a knight, Paola and Minsuk are knaves
G) Paola is a knight, Zoey and Minsuk are knaves
H) Minsuk is a knight, Zoey and Paola are knaves

Case	What is Zoey?	What is Paola?	What is Minsuk?
A	Knight	Knight	Knight
B	Knave	Knave	Knave
C	Knight	Knight	Knave
D	Knight	Knave	Knight
E	Knave	Knight	Knight
F	Knight	Knave	Knave
G	Knave	Knight	Knave
H	Knave	Knave	Knight

The knaves are highlighted in red

If you think of any combination of who is a knight and who is a knave, that combination is listed somewhere in the table.

More generally, for a Knights and Knaves problem, there are 2^N cases, where N is the number of people

For 2 people, $2^2 = 4$ cases
For 3 people, $2^3 = 8$ cases
For 4 people, $2^4 = 16$ cases
and so on ...

Why 2^N and not 3^N or some other number?
Because there are two possibilities for each person: Knight or Knave

Step 4: Put steps 2 and 3 together

(1A) If Kenta is a knight.

then the statement "Reethi is a knave" is true
and Reethi must be a knave

(1B) If Kenta is a knave

then the statement "Reethi is a knave" is a lie
and Reethi must not be a knave
so Reethi must be a knight.

(2A) If Reethi is a knight:

then the statement "Kenta and I are both knights" is true
and Reethi and Kenta are both knights.

(2B) If Reethi is a knave:

then the statement "Kenta and I are both knights" is a lie
and either:
Kenta is a knight and Reethi is a knave
OR Kenta is a knave and Reethi is a knight
OR Kenta and Reethi are both knaves

Case A: Kenta and Reethi are both Knights

Since Case A says Kenta is a knight, we look at (1A)
which tells us that Reethi is a knave.

But Case A says Reethi is a knight

So Case A is impossible.

Case B: Kenta and Reethi are both knaves

Since Case B says Kenta is a knave, we look at (1B)
which tells us that Reethi is a knight

But Case B says Reethi is a knave

So Case B is impossible.

Case C: Kenta is a knight, Reethi is a knave.

Since Case C says Kenta is a knight, we look at (1A)
which tells us that Reethi is a knave.

This is good since Case C says Reethi is a knave

Now let's check what Case C says about Reethi

Since Case C says Reethi is a knave, we look at (2B)
which tells us that either
Kenta is a knight and Reethi is a knave
OR Kenta is a knave and Reethi is a knight
OR Kenta and Reethi are both knaves

This is also good since Case C says
Kenta is a knight and Reethi is a knave

We checked both assumptions of Case C
and nothing went wrong,

So Case C is possible

(next page)

Case D: Kenta is a knave, Reethi is a knight

Since Case D says Kenta is a knave, we look at (1B) which tells us that Reethi is a knight.

This is good since Case D says Reethi is a knave

Now let's check what Case D says about Reethi

Since Case D says Reethi is a knight, we look at (1A) which tells us that Kenta and Reethi are both knights.

But Case D says Kenta is a knave.

So Case D is impossible

As a chart this would be:

Case	Kenta	Reethi	Evaluation
A	Knight	Knight	Impossible
B	Knave	Knave	Impossible
C	Knight	Knave	Possible
D	Knave	Knight	Impossible

Since Case C is the only possible case, we know that

Kenta is a knight and
Reethi is a knave

Boolean Expressions & Truth Tables

In this class, you'll see boolean expressions and truth tables often

Boolean Expression

a symbolic representation of a logic statement

Example

Logic Statement

Paula is NOT a knight AND Zoey is a knight

Boolean expression

$P'Z$ (means the same as the above)

Boolean Variable

a single symbol that represents a TRUE condition

We can represent a simple statement

"Paula is a knight"

with a single Boolean variable P

We can represent the statement's inverse using an apostrophe

So P' is the same as "Paula is not a knight"

This works for any logical statement, not just Knights and Knaves

"It is raining" can be represented by R

Then, R' represents the statement "It is not raining"

This leads us to the concept of Boolean operators.

There are 3 basic Boolean operators: AND, OR, NOT

By discussing inverses, we've essentially discussed NOT

"Paula is not a knight" could be written as

"It is NOT true that Paula is a knight" or even shorter as

NOT P where P represents "Paula is a knight"

We can write NOT P simply as P'

Now, let's consider AND

"Paula is a knight AND Minsuk is a knight"

can be written as P AND M where M represents the statement "Minsuk is a knight"

We can write P AND M simply as $P \cdot M$ or just PM

Finally, let's consider OR

"Paola is a knight OR Minsuk is a knight"

can be written as P OR M

We can write P OR M simply as $P + M$

Logical Operator	Boolean Operator	Meaning
NOT P	P'	P is not true
P AND M	PM	P and M are both true
P OR M	$P + M$	P is true or M is true or P and M are both true

A fourth operator: XOR

So far, we've talked about OR, which is actually inclusive OR

A OR B is true in three cases

A is true and B is false
A is false and B is true
A is true and B is true ← What if we don't want this to be allowed?

Enter "exclusive OR" or XOR

A XOR B is true is true in only two cases

A is true and B is false
A is false and B is true

We can write A XOR B as $A \oplus B$

XOR is technically not a basic gate

This is because we can write XOR using just AND, OR, NOT

Let's think about the two cases when XOR is true

① A is true AND B is false

This is the same as AB'

② A is false AND B is true

This is the same as $A'B$

So A XOR B is true when ① or ② is true

In other words AB' OR $A'B$

which is the same as writing $AB' + A'B$

So, $A \oplus B = \underset{\uparrow}{AB' + A'B}$

But doesn't this technically allow

AB' AND $A'B$ to be true?

Yes. But, think about it.

Is it possible for AB' AND $A'B$ to be true?

If AB' is true then A is true and B is true

Since A is true, $A'B$ can't be true (and vice versa)

So, since AB' AND $A'B$ will never be true, we don't need to worry.

In fact, $AB' \oplus A'B = AB' + A'B$

We'll show this is true in the coming classes.

Truth tables

You've likely seen truth tables before that look something like this

Each row represents one of all of the possible "cases" (or worlds)

A	B	A AND B
False	False	False
False	True	False
True	False	False
True	True	True

On the left, we have a column for each variable (or "input") e.g., A, B

On the right, we have a column for the expression we want to evaluate (or the "output")

Note: The number of possible cases depends on the number of variables (or "inputs") you have

In this example, we have two inputs: A and B

The way to determine the number of cases is by computing 2^N where N is the number of inputs.

In this case: $2^2 = 4$ cases.

Why? (see next page)

An Aside

Where do we get 2^N from?

Well, let's think about each variable we have

A can be true or false (2 possibilities)

B can be true or false (2 possibilities)

Let's fix A to be false

there are two possible cases that B can be (true or false).

① A is false and B is false

② A is false and B is true

If we fix A to be true

there are again two possible cases for B.

③ A is true and B is false

④ A is true and B is true

That's a total of 4 cases, which is $2 \cdot 2$ or 2^2 cases.

We can do the same thing with 3 variables A, B, C

fix A to be false
  two cases for B
    fix B to be false
      two cases for C
        ① A = false, B = false, C = false
        ② A = false, B = false, C = true
    fix B to be true
      two cases for C again
        ③ A = false, B = true, C = false
        ④ A = false, B = true, C = false

(continued on next page)

Now, fix A to be true
since A changed, we have two new cases for B
fix B to be false
two cases for C
⑤ A = True, B = False, C = False
⑥ A = True, B = False, C = True
fix B to be true
two cases for C
⑦ A = True, B = True, C = False
⑧ A = True, B = True, C = True

That's a total of $8 = 2 \cdot 2 \cdot 2 = 2^3$ cases.

You can do this for 4 or more variables as well.

for 4 variables, there are $2^4 = 16$ cases

for 5 variables, there are $2^5 = 32$ cases

... and so on...

Back to truth tables

In this class, we'll write truth tables using 0 and 1.

True/False Version

A	B	A AND B
False	False	False
False	True	False
True	False	False
True	True	True

0/1 Version

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Notice how False is replaced with 0
and True is replaced with 1] Important!

A 3-input (or "3-variable") truth table might look like this:

Notice how the B and C columns repeat when A changes

A	B	C	A AND B AND C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Truth tables for Boolean Operators

We've seen the truth table for AND

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

← this means A AND B

← notice how it's only true when both A and B are true.

The truth table for OR looks like this

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

← this means A OR B

← notice how it's only false when both A and B are false.

The truth table for NOT looks like this:

A	A'
0	1
1	0

← this means NOT A.

Notice how the value is flipped

Also, notice how there's only one variable.

Intermediate columns

Sometimes it's helpful to add columns to your truth tables that aren't inputs or outputs

These columns can represent intermediate stages

For example:

A	B	A'	B'	AB'	$A'B$	$AB' + A'B$
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	0	0	0

← this is XOR $(A \oplus B)$

Notice how the A' column depends on the A column
the B' column depends on the B column
the AB' column depends on the A and B' columns
the $A'B$ column depends on the A' and B columns
the $AB' + A'B$ column depends on the AB' and $A'B$ columns

This example shows us something else.

There's an order of operations

NOT then AND then OR

(more on the next page)

Order of Operations

There's an order in more complex which boolean expressions are evaluated

Example:

$$AB' + A'B$$

Let's say we know the values of A and B.

$$A = 0 \ , B = 1$$

We can plug these values in to our expression

$$(0)(1)' + (0)'(1)$$

The order of operations is NOT then AND then OR

Let's evaluate the NOTs.

$$(0)(0) + (1)(1)$$

Then, we evaluate the ANDs.

$$0 + 1$$

Finally, we evaluate the ORs.

$$1$$

So, if $A = 0$ and $B = 1$, then $A \oplus B = AB' + A'B = 1$

XOR in the order of operations

Sometimes, XOR will be part of your equations

Example

$$A' \oplus B + C$$

In this case, the order of operations becomes:

~~NOT then XOR then AND then OR~~

NOT then AND then XOR then OR

A	B	C	A'	$A' \oplus B$	$A' \oplus B + C$
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	1	0	0
0	1	1	1	0	1
1	0	0	0	0	0
1	0	1	0	0	1
1	1	0	0	1	1
1	1	1	0	1	1

↑ First NOT

↑ Then XOR

↑ Then OR

In this case, we don't have an AND.

Knights and Knaves

Let's consider an example logical statement from Knights and Knaves.

Xochitl is NOT a knave AND
EITHER Minsuk is a knight OR Paola is NOT a knight

First, let's write the statement in terms of only knights

Xochitl is a knight AND
Either Minsuk is a knight OR Paola is NOT a knight

Now, we can this as a logic expression with variables

X AND M XOR NOT P

Now, using the order of operations, we can write it as a boolean expression

X AND M XOR P'

~~X AND $M \oplus P'$~~ → XM XOR P'

~~$X(M \oplus P')$~~ $XM \oplus P'$

~~We write parantheses to show that $M \oplus P'$ happens before and-ing with X.~~

Boolean Algebra: A Companion to Truth Tables

We've talked about truth tables, and we've seen an example of how to show that two Boolean expressions are equal using truth tables.

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

A	B	A'	B'	$A'B$	AB'	$A'B + AB'$
0	0	1	1	0	0	0
0	1	1	0	1	0	1
1	0	0	1	0	1	1
1	1	0	0	0	0	0

These columns are the same.
Therefore,

$$A \oplus B = A'B + AB'$$

You might have noticed that writing truth tables can be tedious (especially if you have 4 or more variables)

↑ that's 16+ columns

So, in these notes, we'll discuss another way to show that two Boolean expressions are equal

Boolean Algebra

To start, let's remember the vocabulary of Boolean expressions (we will use this same vocabulary when writing Boolean algebra equations)

Symbol	Meaning	Example
$\cdot$	AND	$X \cdot Y$ (X AND Y)
$+$	OR	$X + Y$ (X OR Y)
$'$	NOT	X' (NOT X)
$\oplus$	XOR	$X \oplus Y$ (EITHER X OR Y)
0	FALSE	
1	TRUE	
$=$	EQUALS	$X \oplus Y = XY' + X'Y$

↑ when two expressions are the same

Now that we have these symbols, we can start writing some basic rules for Boolean Algebra

Rules for NOT
① $0' = 1$ (NOT FALSE is TRUE)
② $1' = 0$ (NOT TRUE is FALSE)

Rules for AND
③ $0 \cdot 0 = 0$ (FALSE AND FALSE is FALSE)
④ $1 \cdot 1 = 1$ (TRUE AND TRUE is TRUE)
⑤ $0 \cdot 1 = 1 \cdot 0 = 0$ (TRUE AND FALSE is FALSE)
(FALSE AND TRUE is FALSE)

Rules for OR
⑥ $0 + 0 = 0$ (FALSE OR FALSE is FALSE)
⑦ $1 + 1 = 1$ (TRUE OR TRUE is TRUE)
⑧ $0 + 1 = 1 + 0 = 1$ (FALSE OR TRUE is TRUE)
(TRUE OR FALSE is TRUE)

These aren't new rules. We've just written down the rules we use to fill out a truth table, using the symbols of Boolean Algebra.

Now, let's think about what happens when we consider variables. We know that a variable (for example X) has a value of either 0 or 1.

Rules for AND with one variable

What happens if we AND X with 0?

$X \cdot 0$ (same as $0 \cdot X$)

well, there are two cases for what X is.

Let's draw the truth table

This is just a column for the Boolean expression 0. 0 is always 0.

X	0	X·0
0	0	0
1	0	0

These columns are the same.

So, we have another rule.

⑨ $X \cdot 0 = 0$
$0 \cdot X = 0$

What happens if we AND X with 1?

$X \cdot 1$ (same as $1 \cdot X$)

Again, two cases for what X is. Truth table!

This is just a column for the Boolean expression 1.
1 is always 1.

X	1	$X \cdot 1$
0	1	0
1	1	1

These two columns are the same.

So, we have another rule.

⑩ $X \cdot 1 = X$
$1 \cdot X = X$

What if we AND X with itself?

$X \cdot X$

Truth table. Notice, we only have one variable.

X	$X \cdot X$	
0	0	← This is $0 \cdot 0$
1	1	← This is $1 \cdot 1$

These columns are the same. New rule!

⑪ $X \cdot X = X$

What happens if we AND X with its complement?

$X \cdot X'$

One Variable. Two Cases. Truth Table!

X	X'	$X \cdot X'$
0	1	0
1	0	0

This column is always 0. New rule!

⑫ $X \cdot X' = 0$

Rules for OR with one variable

What happens if we OR X with 0?

$X + 0$ (same as $0 + X$)

One variable. Two Cases. Truth Table.

X	0	$X + 0$
0	0	0
1	0	1

The Same! New rule!

⑬ $X + 0 = X$
$0 + X = X$

What happens if we OR X with 1 ?

$X + 1$ (same as $1 + X$)

X	1	X + 1
0	1	1
1	1	1

Same! New Rule!

⑭ $X + 1 = 1$
$1 + X = 1$

What happens when we OR X with itself?

$X + X$

X	X + X	
0	0	← 0 + 0
1	1	← 1 + 1

Same! New Rule!

⑮ $X + X = X$

What happens when we OR X with its complement?

$X + X'$

X	X'	$X + X'$
0	1	1
1	0	1

↓

Always 1. New Rule!

⑯ $X + X' = 1$

Rule for NOT with one variable

What happens if we NOT a variable twice?

$(X')'$

X	X'	$(X')'$
0	1	0
1	0	1

↓

Same! New Rule!

⑰ $(X')' = X$

Great, now we have 9 additional rules that we can use in situations where our Boolean expression has one variable.

Now, what about when the expression has more than one variable.

Rules for more than one variable

When ANDing two variables, does the order matter? Consider two variables X and Y.

X	Y	$X \cdot Y$	$Y \cdot X$
0	0	$0 \cdot 0 = 0$	$0 \cdot 0 = 0$
0	1	$0 \cdot 1 = 0$	$1 \cdot 0 = 0$
1	0	$1 \cdot 0 = 0$	$0 \cdot 1 = 0$
1	1	$1 \cdot 1 = 1$	$1 \cdot 1 = 1$

These results are the same. The order doesn't matter!

⑱ $X \cdot Y = Y \cdot X$ ← New rule!

When ORing two variables, does the order matter?

X	Y	X + Y	Y + X
0	0	$(0+0) = 0$	$(0+0) = 0$
0	1	$(0+1) = 1$	$(1+0) = 1$
1	0	$(1+0) = 1$	$(0+1) = 1$
1	1	$(1+1) = 1$	$(1+1) = 1$

These results are the same. The order doesn't matter!

⑲ $X + Y = Y + X$ ← New rule!

Rules ⑱ and ⑲ tell us that

AND and OR are commutative

(i.e. the order doesn't matter)

(next page)

When ANDing more than two variables, does it matter which AND you do first?

Consider three variables X, Y, and Z.

X	Y	Z	$X\cdot Y$	$(X\cdot Y)\cdot Z$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

These two columns are the same.

The order you evaluate the ANDs does not matter.

X	Y	Z	$Y\cdot Z$	$X\cdot(Y\cdot Z)$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

⑳ $(X\cdot Y)\cdot Z = X\cdot(Y\cdot Z)$

When ORing more than two variables, does it matter which OR you do first?

X	Y	Z	X+Y	(X+Y)+Z
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

These two columns are the same

The order you evaluate the ORs doesn't matter.

X	Y	Z	Y+Z	X+(Y+Z)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

㉑ $(X+Y)+Z = X+(Y+Z)$

Rules ⑳ and ㉑ tell us that AND and OR are each "associative".

Note!

$$X + (Y \cdot Z) \neq (X+Y) \cdot Z$$

X	Y	Z	$Y \cdot Z$	$X + (Y \cdot Z)$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

These columns are not identical.

So they are not equal.

X	Y	Z	$X+Y$	$(X+Y) \cdot Z$
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Remember the Order of Operations

The Distributive property

Is $X \cdot (Y+Z) = (X \cdot Y) + (X \cdot Z)$?

X	Y	Z	$Y+Z$	$X \cdot (Y+Z)$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

These are the same!

So Yes!

X	Y	Z	$X \cdot Y$	$X \cdot Z$	$(X \cdot Y) + (X \cdot Z)$
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	1	1	1

㉒ $X \cdot (Y+Z) = (X \cdot Y) + (X \cdot Z)$

This might look familiar if you saw it in your regular algebra class.

But, what about this?

Is $X + (Y \cdot Z) = (X+Y) \cdot (X+Z)$?

X	Y	Z	$Y \cdot Z$	$X + (Y \cdot Z)$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

These are the same!

So Yes!

X	Y	Z	$X+Y$	$X+Z$	$(X+Y) \cdot (X+Z)$
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

㉓ $X + (Y \cdot Z) = (X+Y) \cdot (X+Z)$

If this looks weird to you, that might be because in standard algebra this equation would not be true. In Boolean algebra, it is.

DeMorgan's Rules

The last two rules we'll talk about are:

$$(X \cdot Y)' = X' + Y'$$

X	Y	$X \cdot Y$	$(X \cdot Y)'$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

These are the same!

X	Y	X'	Y'	$X' + Y'$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

㉔ $(X \cdot Y)' = X' + Y'$

and $(X + Y)' = X' \cdot Y'$

(see next page for truth tables)

$(x+y)' = x' \cdot y'$

x	y	x + y	(x + y)'
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

These are the same

x	y	x'	y'	x' · y'
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

㉕ $(x+y)' = x' \cdot y'$

Okay, so 25 rules! Sounds like a lot, but the first 8 are just what we use to fill out a truth table, and the rest can be combined somewhat

Recap of Rules

① $0' = 1$
② $1' = 0$

This is what NOT does

③ $0 \cdot 0 = 0$
④ $1 \cdot 1 = 1$
⑤ $0 \cdot 1 = 1 \cdot 0 = 0$

This is what AND does

⑥ $0 + 0 = 0$
⑦ $1 + 1 = 1$
⑧ $0 + 1 = 1 + 0 = 1$

This is what OR does

⑨ $0 \cdot X = X \cdot 0 = 0$
⑩ $1 \cdot X = X \cdot 1 = X$
⑪ $X \cdot X = X$
⑫ $X \cdot X' = 0$

Rules for one variable AND

⑬ $0 + X = X + 0 = X$
⑭ $1 + X = X + 1 = 1$
⑮ $X + X = X$
⑯ $X + X' = 1$

Rules for one variable OR

⑰ $(X')' = X$ ← Double Complement
(Double NOT)

(next page)

⑱	$X \cdot Y = Y \cdot X$	Commutative
⑲	$X + Y = Y + X$	
⑳	$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$	Associative
㉑	$(X + Y) \cdot Z = X + (Y \cdot Z)$	
㉒	$X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$	Distributive
㉓	$X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$	
㉔	$(X \cdot Y)' = X' + Y'$	DeMorgan's
㉕	$(X + Y)' = X' \cdot Y'$	

So, you might notice that the last eight rules are really only four, each applied to AND, then OR.

	AND		OR
Commutativity	⑱	and	⑲
Associativity	⑳	and	㉑
Distributivity	㉑	and	㉒
DeMorgan's	㉔	and	㉕

The first 8 are just how AND, OR, NOT work.

Let's see if we can group the other 9.

"Null Element"

⑨ $0 \cdot x = x \cdot 0 = 0$ (AND)

⑭ $1 + x = x + 1 = 1$ (OR)

These are both rules where the value of x doesn't affect the result.

"Identity"

⑩ $1 \cdot x = x \cdot 1 = x$ (AND)

⑬ $0 + x = x + 0 = x$ (OR)

These are both rules where the result is just the value of x.

"Idempotent"

⑪ $x \cdot x = x$ (AND)

⑮ $x + x = x$ (OR)

These are both rules that let you remove repeated terms.

"Inverse"

⑫ $x \cdot x' = 0$ (AND)

⑯ $x + x' = 0$ (OR)

These are both rules

"Double complement"

⑰ $(X')' = X$ (NOT)

You can think of this rule as toggling a light switch. If the switch is off : X', then toggling it $(X')'$ turns it back on : X.

So, summing this up, we have the following types of rules

Category	Rules related to AND	Rules related to OR	Rules related to NOT
Basic Definitions	③ through ⑤	⑥ through ⑧	① and ②
Null Element	⑨	⑭	NA
Identity	⑩	⑬	NA
Idempotent	⑪	⑮	NA
Inverse	⑫	⑯	
Double Complement	NA	NA	⑰
Commutativity	⑱	⑲	NA
Associativity	⑳	㉑	NA
Distributive	㉒	㉓	NA
DeMorgan's	㉔	㉕	

Category	Rules related to AND	Rules related to OR	Rules related to NOT
Basic Definitions	$0 \cdot 0 = 0$ $1 \cdot 1 = 1$ $0 \cdot 1 = 1 \cdot 0 = 0$	$0 + 0 = 0$ $1 + 1 = 1$ $0 + 1 = 1 + 0 = 0$	$0' = 1$ $1' = 0$
Null Element	$0 \cdot x = x \cdot 0 = 0$	$1 + x = x + 1 = 1$	
Identity	$1 \cdot x = x \cdot 1 = x$	$0 + x = x + 0 = x$	
Idempotent	$x \cdot x = x$	$x + x = x$	
Inverse	$x \cdot x' = 0$	$x + x' = 1$	
Double Complement			$(x')' = x$
Commutativity	$x \cdot y = y \cdot x$	$x + y = y + x$	
Associativity	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	$(x + y) + z = x + (y + z)$	
Distributive	$x \cdot (y + z) = xy + xz$	$x + (y \cdot z) = (x + y) \cdot (y + z)$	
DeMorgan's	$(x \cdot y)' = x' + y'$	$(x + y)' = x' \cdot y'$	

Many of these rules are the same as the rules from standard algebra.

The ones that are unique to Boolean algebra are highlighted in red

Now that we have the rules, we can use them.
This part will take time to develop an intuition
The more you see and do Boolean algebra problems
the quicker you'll get

Let's start with a Boolean algebra equation
that we've seen before
Let's prove that $(a+b)\cdot(a+c) = a+(b\cdot c)$
This is just rule ㉓ distributivity for OR.

We want to get from $(a+b)\cdot(a+c)$ to $a+(b\cdot c)$
using just the rules we know
(not including rule ㉓)

(Boolean algebra proof on next page)

$$(a+b)\cdot(a+c) = a+(b\cdot c)$$

Proof:

First, we'll use rule ㉒ to distribute.
This works the same way as it does with
standard algebraic expressions

I'm omitting the $\cdot$ and ( ) where possible.

$$(a+b)\,(a+c) = aa + ac + ba + bc$$

Now, we can use rule ⑪ to replace aa

$$= a + ac + ba + bc$$

Now, use rule ⑩ to replace a with $a \cdot \underline{1}$
and rule ⑱ to flip ba.

$$= a\cdot 1 + ac + ab + bc$$

Now, use rule ㉒ to factor out a from
the first 3 terms.

$$= a(1 + c + b) + bc$$

Now, use rule ⑭ twice to remove c+b.

$$= a(1) + bc$$

Finally, use rule ⑩ to remove the 1.

$$= a + bc$$

$(a+b)(a+c) = aa + ac + ba + bc$ (distributivity of AND) rule ㉒

$= a + ac + ba + bc$ (idempotent of AND) rule ⑪

$= a \cdot 1 + ac + ba + bc$ (identity of AND) rule ⑩

$= a \cdot 1 + ac + ab + bc$ (commutativity of AND) rule ⑧

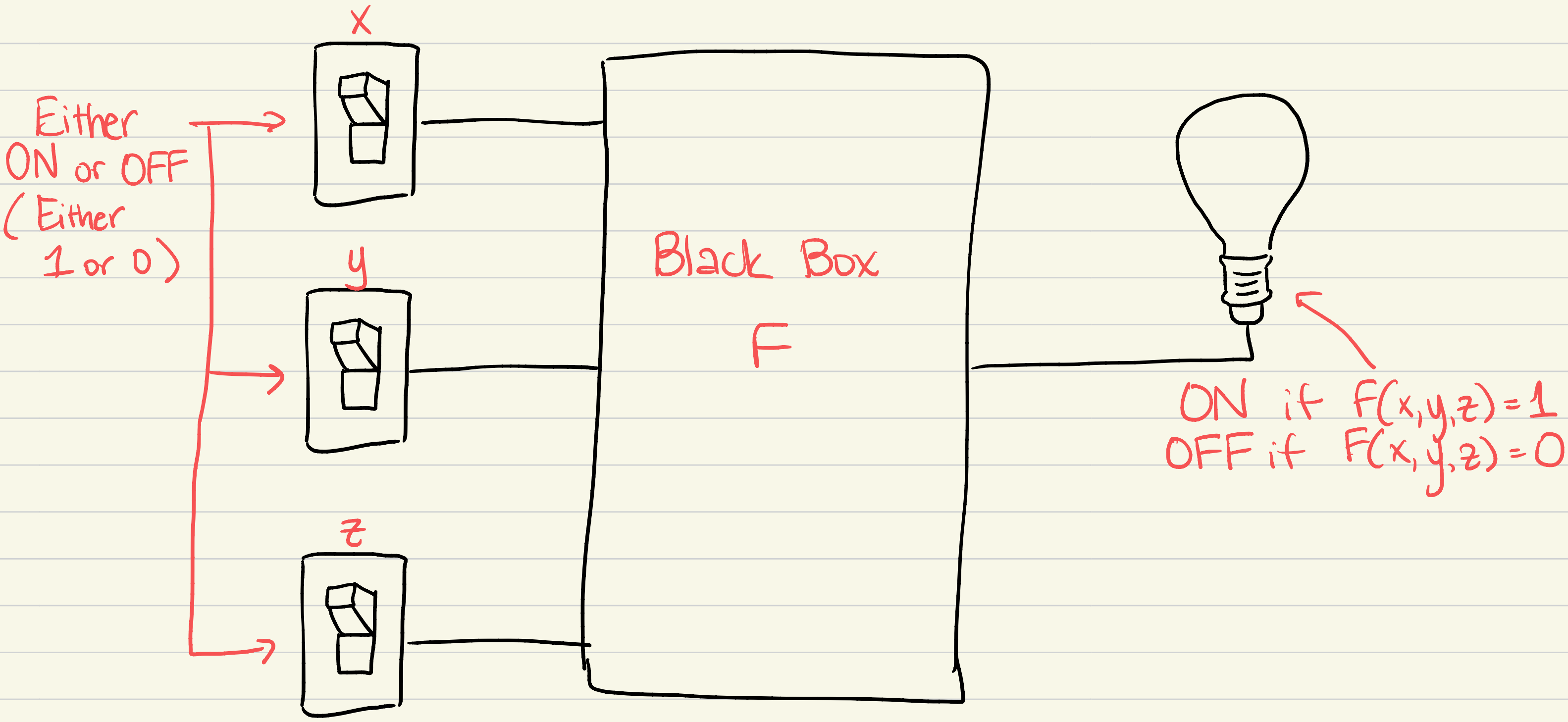
$= a(1 + c + b) + bc$ (distributivity of AND) rule ㉒

$= a(1) + bc$ (null element of OR) rule ⑨

$= a + bc$ (identity of AND) rule ⑩

Deriving Boolean equations from truth tables

- When designing circuits, it is common to be given only a truth table of how a circuit should work


x
y
z
Either ON or OFF (Either 1 or 0)
Black Box
F
ON if F(x,y,z)=1
OFF if F(x,y,z)=0

For example, we might be given the truth table for the black box "F" above

0 means the lightswitch is "OFF"

1 means the lightswitch is "ON"

x	y	z	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

1 means the lightbulb is ON

0 means the lightbulb is OFF

SUM-OF-PRODUCTS FORM

That truth table actually holds all the information we need to write a Boolean expression for the black box F.

One way to do this is using the "sum-of-products" approach. Let's define a few terms.

Product term

- A Boolean expression that only contains variables, NOT, and AND.
- Examples:
  - → xyz
  - → a'
  - → $p'qrs'$
- $(x + y'z + w)$ is not a product term because it contains ORs (+)

Sum-of-Products (SOP)

- A sum-of-products is a Boolean expression where multiple product terms are OR'd (summed) together.
- Example:

$$\underbrace{abc' + ac + ab'}_{\text{3 product terms}}$$

- $[abc' + a(c+b')]$ is not in sum-of-products form because $a(c+b')$ is not a product term (it contains OR)

Canonical Sum-of-Products

- A "canonical" sum-of-products is a sum-of-products where each product term has every variable that shows up in the whole Boolean expression
- Example:

$$\underbrace{x'yz' + x'yz + xyz}_{\text{Each product term has all 3 variables } x, y, z.}$$

Note: each product term has x or x', (not both)
y or y'
z or z'

- $xyz + x'z$ is not a canonical sum-of-products

$\uparrow$

this product term is missing y or y'.

- A product term with all of the variables in a Boolean expression is called a **minterm**

Now that we know what a canonical sum-of-products is, let's go back to the truth table for F that we were looking at before:

x	y	z	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Given a truth table for F, we can write a Boolean equation for F in sum-of-products form.

(see next page)

x	y	z	F	
0	0	0	1	Ⓐ
0	0	1	0	
0	1	0	1	Ⓑ
0	1	1	1	Ⓒ
1	0	0	1	Ⓓ
1	0	1	0	
1	1	0	1	Ⓔ
1	1	1	0	

To do this, let's consider a single row of the truth table where $F = 1$. (the first row).

(I've labeled all of the rows where $F = 1$)

Now let's consider a truth table where Ⓐ was the only row where $F = 1$.

x	y	z	F_A	
0	0	0	1	Ⓐ
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	0	
1	1	1	0	

What would the boolean expression for F_A be?

Well F_A is only 1 when $x = 0$ AND $y = 0$ AND $z = 0$

So, we can say $F_A = x'y'z'$

(next page)

Let's double check that $F = x'y'z'$ using truth tables.

x	y	z	x'	y'	z'	$x'y'z'$ ✓
0	0	0	1	1	1	1
0	0	1	1	1	0	0
0	1	0	1	0	1	0
0	1	1	1	0	0	0
1	0	0	0	1	1	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	0

← this column is the same as F.

So, $F = x'y'z'$.

Good!

In fact, we could do this for each row where F was originally 1.

(next page)

x	y	z	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

This is what F was orignally

x	y	z	F_B	
0	0	0	0	
0	0	1	0	
0	1	0	1	Ⓑ
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	0	
1	1	1	0	

F_B is only 1 when

$x=0$ AND $y=1$ AND $z=0$

So $F_B = x'yz'$

x	y	z	F_C	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	Ⓒ
1	0	0	0	
1	0	1	0	
1	1	0	0	
1	1	1	0	

F_C is only 1 when

$x=0$ AND $y=1$ AND $z=1$

So $F_C = x'yz$

(next page)

x	y	z	F_D
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Ⓓ

F_D is only 1 when

$x = 1$ AND $y = 0$ AND $z = 0$

So $F_C = xy'z'$

x	y	z	F_E
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

F_E is only 1 when

$x = 1$ AND $y = 1$ AND $z = 0$

So $F_E = xyz'$

More generally, each row of a truth table can be written as a Boolean expression with a single minterm.

Now, let's put these five truth tables together to show how we can get the Boolean equation for F.

x	y	z	F_A	F_B	F_C	F_D	F_E	F
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	0	0
0	1	0	0	1	0	0	0	1
0	1	1	0	0	1	0	0	1
1	0	0	0	0	0	1	0	1
1	0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	1	1
1	1	1	0	0	0	0	0	0

Notice a pattern?

x	y	z	F_A	F_B	F_C	F_D	F_E	$F_A + F_B + F_C + F_D + F_E$
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	0	0
0	1	0	0	1	0	0	0	1
0	1	1	0	0	1	0	0	1
1	0	0	0	0	0	1	0	1
1	0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	1	1
1	1	1	0	0	0	0	0	0

$F = F_A + F_B + F_C + F_D + F_E$

F is just the columns for F_A through F_E OR'd together.

We know that $F_A = x'y'z'$; $F_B = x'yz'$

$F_C = x'yz$; $F_D = xy'z'$

and $F_E = xyz'$

So, $F = x'y'z' + x'yz' + x'yz + xy'z' + xyz'$

Generally, given a truth table F, we can write the Boolean function for F by summing (ORing) the minterms for the rows where F=1.

Here is a table of minterms for 3 variables

A	B	C	minterm	minterm #
0	0	0	A'B'C'	0
0	0	1	A'B'C	1
0	1	0	A'BC'	2
0	1	1	A'BC	3
1	0	0	AB'C'	4
1	0	1	AB'C	5
1	1	0	ABC'	6
1	1	1	ABC	7

Here is a table of minterms for 4 variables

A	B	C	D	minterm	minterm #
0	0	0	0	A'B'C'D'	0
0	0	0	1	A'B'C'D	1
0	0	1	0	A'B'CD'	2
0	0	1	1	A'B'CD	3
0	1	0	0	A'BC'D'	4
0	1	0	1	A'BC'D	5
0	1	1	0	A'BCD'	6
0	1	1	1	A'BCD	7
1	0	0	0	AB'C'D'	8
1	0	0	1	AB'C'D	9
1	0	1	0	AB'CD'	10
1	0	1	1	AB'CD	11
1	1	0	0	ABC'D'	12
1	1	0	1	ABC'D	13
1	1	1	0	ABCD'	14
1	1	1	1	ABCD	15

Another example:

inputs			output (think of this guy as F)	minterms
☺	□	△	◇	
0	0	0	1	☺'□'△'
0	0	1	0	
0	1	0	0	
0	1	1	1	☺'□△
1	0	0	0	
1	0	1	0	
1	1	0	1	☺□△'
1	1	1	0	

Here we want to find a Boolean equation for "◇" in terms of ☺, □, and △.

Looking at the rows with ◇ = 1,

we have the 3 minterms:

☺'□'△', ☺'□△, and ☺□△'

OR all the minterms together and you get:

◇ = ☺'□'△' + ☺'□△ + ☺□△'

PRODUCT-OF-SUMS FORM

Another way to derive a Boolean equation from a truth table is using the "product-of-sums" approach. Let's define a few more terms.

Sum Term

A Boolean expression that contains only variables, NOT, and OR

- Examples:
  - → $(x + y' + z)$
  - → $(a + b + c')$
  - → $(p + q)$
- $p + qr'$ is not a Sum term because it contains AND $(\cdot)$.

So, $F(x,y,z) = (x+y+z)(x+y'+z')(x'+y+z)(x'+y'+z)(x'+y'+z')$

is a Product-of-Sums

Product-of-Sums

- A Boolean expression where multiple Sum terms are AND'd together
- Examples:
  - → $(a+b)(c'+d)$
  - → $(p+q'+r)(s'+q)$

- $(a+b)(a'+bc)$ is not in product-of-sums form because $(a'+bc)$ is not a sum term (it contains AND).

Canonical Product-of-sums

- A "canonical" product-of-sums is a product-of-sums where each sum term has every variable that shows up in the whole Boolean expression
- Example:

$$(x+y+z)(x+y'+z')(x'+y+z)(x'+y'+z)(x'+y'+z')$$

Each sum term has all 3 variables x, y, z.

Note: each sum term has x or x', (not both)
y or y'
z or z'

- $(a+b)(c'+d)$ is not a "canonical" product-of-sums because each sum term does not have all 4 variables.
- a "canonical" sum term is called a maxterm

Given a truth table for F, we can also write a Boolean equation for F in product-of-sums form.

Let's use the same truth table for F as an example.

x	y	z	F	
0	0	0	1	
0	0	1	0	Ⓐ
0	1	0	1	
0	1	1	1	
1	0	0	1	
1	0	1	0	Ⓑ
1	1	0	1	
1	1	1	0	Ⓒ

To do this, we'll consider each row of the table where $F=0$

(e.g. row Ⓐ).

Now let's see a truth table where row A is the only row where $F=0$

(next page)

x	y	z	F_A
0	0	0	1
0	0	1	0 Ⓐ
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

What would the Boolean expression for F_A be?

Well $F_A = 0$ when $x = 0$ AND $y = 0$ AND $z = 1$.

This can be represented by $x'y'z$

But, we want to know when $F_A = 1$.

Well $F_A = 1$ when F_A is not 0.

So $F_A = (x'y'z)'$

$= \underbrace{x + y + z'}_{\text{Notice: this is a maxterm}}$

use DeMorgan's rule

Let's double check that with a truth table.

x	y	z	z'	x+y	x+y+z'
0	0	0	1	0	1
0	0	1	0	0	0
0	1	0	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	0	1	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1

This is the same as FA.

Good!

In fact, we can do this for each row where F was originally 0.

(next page)

x	y	z	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

This is what F was originally.

x	y	z	F_B	
0	0	0	1	
0	0	1	1	
0	1	0	1	
0	1	1	1	
1	0	0	1	
1	0	1	0	Ⓑ
1	1	0	1	
1	1	1	1	

$F_B = 0$ when

$x = 1$ AND $y = 0$ AND $z = 1$ → $xy'z$

So, $F_B = (xy'z)'$

$$= \underbrace{x' + y + z'}_{\text{a maxterm}}$$

x	y	z	F_C	
0	0	0	1	
0	0	1	1	
0	1	0	1	
0	1	1	1	
1	0	0	1	
1	0	1	1	
1	1	0	1	
1	1	1	0	Ⓒ

$F_C = 0$ when

$x = 1$ AND $y = 1$ AND $z = 1$ → xyz

So $F_C = (xyz)'$

$$F_C = \underbrace{x' + y' + z'}_{\text{a maxterm}}$$

More generally, each row of a truth table can be written as a Boolean expression with a single maxterm.

Now, let's put these three truth tables together to show how we can get an equation for F.

x	y	z	F_A	F_B	F_C	F
0	0	0	1	1	1	1
0	0	1	0	1	1	0
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	1	1	1
1	0	1	1	0	1	0
1	1	0	1	1	1	1
1	1	1	1	1	0	0

Notice a pattern?

x	y	z	F_A	F_B	F_C	$F_A \cdot F_B \cdot F_C$
0	0	0	1	1	1	1
0	0	1	0	1	1	0
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	1	1	1
1	0	1	1	0	1	0
1	1	0	1	1	1	1
1	1	1	1	1	0	0

(next page)

F is just the result of ANDing F_A, F_B, and F_C.

$$F = F_A \cdot F_B \cdot F_C = \overset{1}{\underset{\overset{\uparrow}{F_A}}{(x+y+z')}} \cdot \overset{5}{\underset{\overset{\uparrow}{F_B}}{(x'+y+z')}} \cdot \overset{7}{\underset{\overset{\uparrow}{F_C}}{(x'+y'+z')}}$$

So, we have the product-of-sums form of F:

$$F(x,y,z) = (x+y+z')(x'+y+z')(x'+y'+z')$$

And we got the sum-of-products form of F before:

$$F(x,y,z) = x'y'z' + x'yz' + x'yz + xy'z' + xyz'$$

(next page)

We can show that these are equivalent using Boolean alg:

$F(x,y,z) = (x+y+z')(x'+y+z')(x'+y'+z')$ using Distributivity

$= (\underset{0}{\cancel{xx'}} + xy + xz' + yx' + \overset{y}{\cancel{yy}} + yz' + z'x' + \cancel{z'y} + \overset{z'}{\cancel{z'z'}})(x'+y'+z')$ using inverse and idempotent

factor out z'

$= (xy + xz' + yx' + yz' + z'x' + z')(x'+y'+z')$ using Distributivity

factor out y

$= (z'(x+x'+1) + y(x+x'+z'))(x'+y'+z')$ using null element, identity, and inverse

$= (z'+y)(x'+y'+z')$ Using Distributivity

$= z'x' + z'y' + \overset{z'}{\cancel{z'z'}} + yx' + \underset{0}{\cancel{yy'}} + yz'$ Using inverse and idempotent

$= z'x' + z'y' + z' + yx' + yz'$ using Distributivity

factor out z'

$= z'(x'+y'+1+y) + yx'$ using null element and identity

$= z' + yx'$ using identity and inverse

$= z'(x+x')(y+y') + yx'(z+z')$ Using distributivity

$= z'xy + z'xy' + z'x'y + z'x'y' + yx'z + \cancel{yx'z'}$ using identity and commutativity

$= xyz' + xy'z' + x'yz' + x'y'z' + x'yz$ Using commutativity

$= x'y'z' + x'yz' + x'yz + xy'z' + xyz'$ ← sum-of-products form.

Generally, given a truth table F, we can write the Boolean function for F by multiplying (ANDing) the maxterms for the rows where F = 0.

Here is a table of maxterms for 3 variables

A	B	C	maxterm	maxterm #
0	0	0	$A+B+C$	0
0	0	1	$A+B+C'$	1
0	1	0	$A+B'+C$	2
0	1	1	$A+B'+C'$	3
1	0	0	$A'+B+C$	4
1	0	1	$A'+B+C'$	5
1	1	0	$A'+B'+C$	6
1	1	1	$A'+B'+C'$	7

Here is a table of maxterms for 4 variables

A	B	C	D	maxterm	maxterm #
0	0	0	0	$A+B+C+D$	0
0	0	0	1	$A+B+C+D'$	1
0	0	1	0	$A+B+C'+D$	2
0	0	1	1	$A+B+C'+D'$	3
0	1	0	0	$A+B'+C+D$	4
0	1	0	1	$A+B'+C+D'$	5
0	1	1	0	$A+B'+C'+D$	6
0	1	1	1	$A+B'+C'+D'$	7
1	0	0	0	$A'+B+C+D$	8
1	0	0	1	$A'+B+C+D'$	9
1	0	1	0	$A'+B+C'+D$	10
1	0	1	1	$A'+B+C'+D'$	11
1	1	0	0	$A'+B'+C+D$	12
1	1	0	1	$A'+B'+C+D'$	13
1	1	1	0	$A'+B'+C'+D$	14
1	1	1	1	$A'+B'+C'+D'$	15

Notice how, for maxterms,

an unprimed variable like B represents $B=0$, while

a primed variable like B' represents $B=1$.

(This is because of DeMorgan's rule)

Another example:

inputs: ☺, □, △

output ↓ (think of this guy as F): ◇

maxterms ↓

☺	□	△	◇		maxterms
0	0	0	1		
0	0	1	0	←	☺ + □ + △'
0	1	0	0	←	☺ + □' + △
0	1	1	1		
1	0	0	0	←	☺' + □ + △
1	0	1	0	←	☺' + □ + △'
1	1	0	1		
1	1	1	0	←	☺' + □' + △'

Here we want to find a Boolean equation for "◇" in terms of ☺, □, and △.

Looking at the rows with ◇ = 0.

we have the 5 maxterms:

☺ + □ + △', ☺ + □' + △, ☺' + □ + △,

☺' + □ + △', and ☺' + □' + △'

AND all the maxterms together and you get:

◇ = (☺ + □ + △')(☺ + □' + △)(☺' + □ + △)(☺' + □ + △')(☺' + □' + △')

Inverse Functions

A function $F(x,y,z)$ has inputs x, y, and z, and its output can be written as a Boolean expression. For example:

$$F(x,y,z) = \underbrace{x + yz'}_{\text{output}}$$

The inverse of $F(x,y,z)$ can be written as:

$$F'(x,y,z) = (F(x,y,z))'$$

or more simply:

$$F' = (F)'$$

For example:

$$\begin{aligned} F'(x,y,z) &= (F(x,y,z))' \\ &= (x + yz')' \\ &= x'(y' + z) \end{aligned}$$

using DeMorgan's rule

In a truth table,

								$x+yz'$	$x'(y'+z)$
x	y	z	x'	y'	z'	yz'	$y'+z$	$F(x,y,z)$	$F'(x,y,z)$
0	0	0	1	1	1	0	1	0	1
0	0	1	1	1	0	0	1	0	1
0	1	0	1	0	1	1	0	1	0
0	1	1	1	0	0	0	1	0	1
1	0	0	0	1	1	0	1	1	0
1	0	1	0	1	0	0	1	1	0
1	1	0	0	0	1	1	0	1	0
1	1	1	0	0	0	0	1	1	0

Purple columns are related to F

Blue columns are related to F'

Notice that the column for F' is just the column for F but NOT'd.

Now let's use this truth table to derive the SOP and POS forms of F as well as the SOP and POS forms of F'

(next page)

x	y	z	F(x,y,z)	F'(x,y,z)
0	0	0	0	1
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

SOP: $F(x,y,z) = \overset{0}{x'}\overset{1}{y}\overset{0}{z'} + \overset{1}{x}\overset{0}{y'}\overset{0}{z'} + \overset{1}{x}\overset{0}{y'}\overset{1}{z} + \overset{1}{x}\overset{1}{y}\overset{0}{z'} + \overset{1}{x}\overset{1}{y}\overset{1}{z}$ ← minterms

POS: $F(x,y,z) = (\overset{0}{x}+\overset{0}{y}+\overset{0}{z})\cdot(\overset{0}{x}+\overset{0}{y}+\overset{1}{z'}) + (\overset{0}{x}+\overset{1}{y'}+\overset{1}{z'})$ ← maxterms

SOP: $F'(x,y,z) = \overset{0}{x'}\overset{0}{y'}\overset{0}{z'} + \overset{0}{x'}\overset{0}{y'}\overset{1}{z} + \overset{0}{x'}\overset{1}{y}\overset{1}{z}$ ← minterms

maxterms

POS: $F'(x,y,z) = (\overset{0}{x}+\overset{1}{y'}+\overset{0}{z})\cdot(\overset{1}{x'}+\overset{0}{y}+\overset{0}{z})\cdot(\overset{1}{x'}+\overset{0}{y}+\overset{1}{z'})\cdot(\overset{1}{x'}+\overset{1}{y'}+\overset{0}{z})\cdot(\overset{1}{x'}+\overset{1}{y'}+\overset{1}{z'})$

From this, notice that the POS form of F' is just the inverse of the SOP form of F

Also notice that the SOP form of F' is just the inverse of the POS form of F

Number Systems

At the fundamental level, computers only have:

two values: 0 and 1

and three operations: AND, OR, NOT.

With these basic tools, we've been able to start representing logical expressions:

e.g., we've learned how to express statements like:

"Either Minsuk is NOT a knight OR Paola is a knight AND Zoey is a knight."

as Boolean formulas:

$$F(M,P,Z) = M' \oplus PZ$$

and as truth tables:

M	P	Z	$F = M' \oplus PZ$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

So, we've got a way to represent logical expressions
using 0, 1, AND, OR, NOT.

But if you remember back to CSC 103, we know
that computers often also need to represent / compute
arithmetic expressions like:

$$215 + 14 \quad 7$$

note: I mean addition here (not OR)

You might have noticed a couple of issues:

We don't have an operator for addition

(we'll talk about this later)

But, more importantly, we don't have any numbers
except for 0 and 1.

How do we represent 215 and 14 with
just 1s and 0s?

This is where "binary" comes in.

But, before we talk about Binary, let's start with the numerical representation you're likely more familiar with: Decimal

Decimal

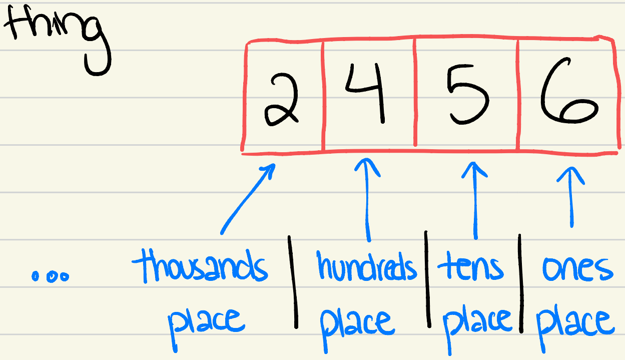
Decimal numbers are just the numbers we typically think of when we think of numbers.

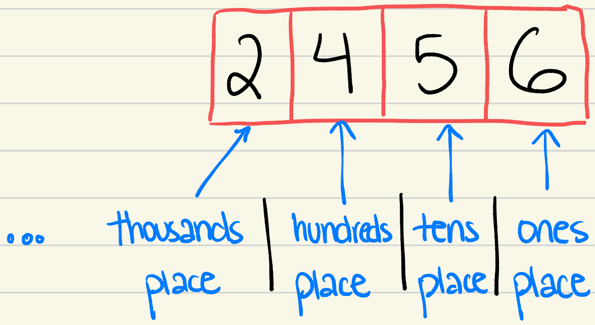
For example: 2456, 23, 37

Let's take a look at how a decimal number is written. We'll use the number 2456 as an example:

2	4	5	6

2456 is comprised of 4 digits. The position of each digit tells you something


2
4
5
6
... thousands place
hundreds place
tens place
ones place



The 6 in the first position (starting from the right) should be interpreted as "six ones" or just "six"

The 5 in the second position (starting from the right) should be interpreted as "five tens" or just "fifty"

The 4 in the third position (starting from the right) should be interpreted as "four hundreds" or just "four hundred"

The 2 in the fourth position (starting from the right) should be interpreted as "two thousands" or just "two thousand"

So, we have, in all, two thousand four hundred fifty six which is what 2456 represents.

There is implied multiplication and addition happening here:

six times one		$(6 \cdot 1)$		6
plus five times ten	same as	$+ (5 \cdot 10)$	same as	$+ 50$
plus four times one hundred		$+ (4 \cdot 100)$		$+ 400$
plus two times one thousand		$+ (2 \cdot 1000)$		$+ 2000$
				2456

Notice that we're multiplying by the next power of 10 each time (starting from $10^0 = 1$)

$$
\begin{matrix}
(6 \cdot 1) & & (6 \cdot 10^0) \\
+ (5 \cdot 10) & \xrightarrow{\text{same as}} & + (5 \cdot 10^1) \\
+ (4 \cdot 100) & & + (4 \cdot 10^2) \\
+ (2 \cdot 1000) & & + (2 \cdot 10^3)
\end{matrix}
$$

So, we can think of 2456 as:

$$
\begin{array}{c|c|c|c|c|}
 & 2 & 4 & 5 & 6 \\
 & \uparrow & \uparrow & \uparrow & \uparrow \\
\cdots & 10^3 & 10^2 & 10^1 & 10^0
\end{array}
$$

$$
\left.\begin{array}{l}
6 \text{ in the } 10^0 \text{ place} \\
5 \text{ in the } 10^1 \text{ place} \\
4 \text{ in the } 10^2 \text{ place} \\
1 \text{ in the } 10^3 \text{ place}
\end{array}\right\}
$$

since all of these exponents have 10 in the base, we call decimal a base-10 system for writing numbers

Now, let's think about how we count in decimal

"zero"	→	0	0	0	0
"one"	→	0	0	0	1
"two"	→	0	0	0	2
"three"	→	0	0	0	3
"four"	→	0	0	0	4
"five"	→	0	0	0	5
"six"	→	0	0	0	6
"seven"	→	0	0	0	7
"eight"	→	0	0	0	8
"nine"	→	0	0	0	9
"ten"	→	0	0	1	0

- In decimal, we have ten unique symbols:

  0 1 2 3 4 5 6 7 8 9

- We start on the right in the 10^0 (ones) place and we count through the symbols from 0 to 9.

- Once we get to 9, we don't have any more symbols left, so we start over at "0" in the 10^0 (ones) place and we move the 10^1 (tens) place up one symbol (digit).

"ten" → | 0 | 0 | 1 | 0 |

"eleven" → | 0 | 0 | 1 | 1 |

... ...

"nineteen" → | 0 | 0 | 1 | 9 |

"twenty" → | 0 | 0 | 2 | 0 |

- Then again we cycle through the digits in the 10^0 (ones) place until we get back to "9"
- Then we go back to "0" and move the 10^1 (tens) place up by 1.

... ...

"ninety-nine" → | 0 | 0 | 9 | 9 |

"one hundred" → | 0 | 1 | 0 | 0 |

- We keep cycling the 10^0 (ones) place and incrementing the 10^1 (ones) place until we get to "99". At this point we have to cycle both the 10^0 (ones) and 10^1 (tens) place back to zero and then move the 10^2 (hundreds) place up 1.

Recap

So, decimal is a base-10 system, which means:

- there are 10 digits (0 through 9) that can go in each position
- each digit d in a decimal number represents $d \cdot 10^i$ (i is the position starting at 0 on the right)

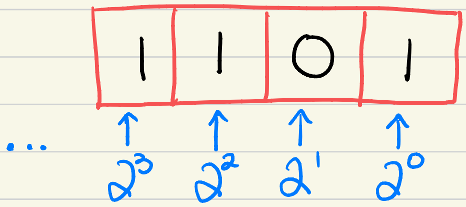
A computer only knows the digits 0 and 1.
"on or off"

That's two digits, so a base-2 number-writing system might come in handy.

Now, for real, here's binary.

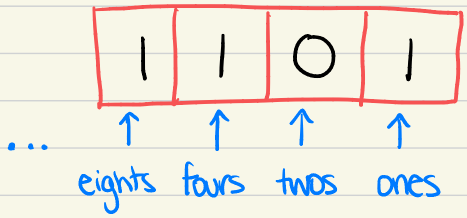
(next page)

The binary number system is structurally very similar to decimal.


1
1
0
1
...
2^3
2^2
2^1
2^0

The position of the digits still starts from 0 on the right and increases by 1 each time we move to the left (this is the exponent in blue)

but now the base of the number system is 2.


1
1
0
1
...
eights
fours
twos
ones

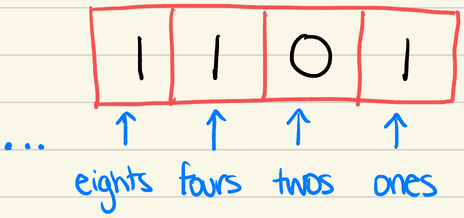
Note: the powers of 2 are:
1, 2, 4, 8, 16, 32, 64, 128, 256, etc.

So instead of the

ones, tens, hundreds, thousands, ... places,

we have the ones, twos, fours, eights, ... places

Thus, interpreting a binary number can be done in a similar way as interpreting a decimal number



The 1 in the ones place can be interpreted as "one one" or just "one"

The 0 in the twos place can be interpreted as "zero twos" or just "zero"

The 1 in the fours place can be interpreted as "one four" or just "four"

The 1 in the eights place can be interpreted as "one eight" or just "eight"

Again there's implied multiplication and addition:

one times one
plus zero times two
plus one times four
plus one times eight

same as

$$\begin{array}{r} (1 \cdot 1) \\ +(0 \cdot 2) \\ +(1 \cdot 4) \\ +(1 \cdot 8) \end{array}$$

same as

$$\begin{array}{r} 1 \\ +0 \\ +4 \\ \underline{+8} \\ 13 \end{array}$$

so this number is representing "thirteen" ↗

Notice that we're multiplying by the next power of 2 each time (starting from the left)

$$\begin{array}{lcl} (1\cdot 1) & & (1\cdot 2^0)\\ +(0\cdot 2) & \xrightarrow{\text{same as}} & +(0\cdot 2^1)\\ +(1\cdot 4) & & +(1\cdot 2^2)\\ +(1\cdot 8) & & +(1\cdot 2^3) \end{array}$$

1	1	0	1
2^3	2^2	2^1	2^0

remember this?

position goes from right to left

So the binary number 1101 =

here we're adding the digits in each position from left to right

$$(1\cdot 2^0)+(0\cdot 2^1)+(1\cdot 2^2)+(1\cdot 2^3)$$

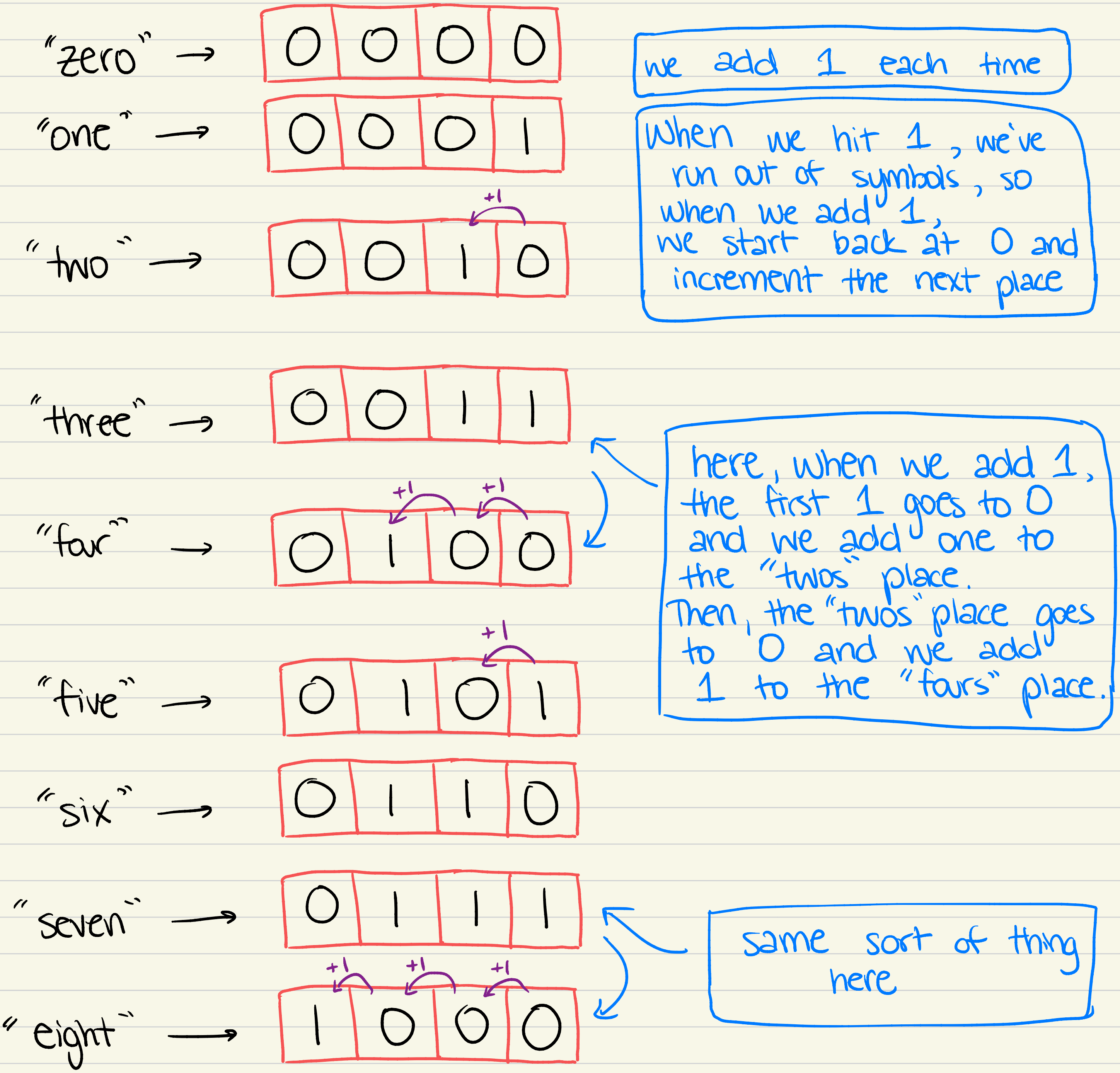
$$=(1\cdot 1)+(0\cdot 2)+(1\cdot 4)+(1\cdot 8)$$

$$=1+0+4+8$$

$$=13$$

Counting in Binary

The process of counting in binary is the same, we just have fewer symbols (only 0 and 1)


"zero" → 0 0 0 0
"one" → 0 0 0 1
"two" → 0 0 1 0
+1
"three" → 0 0 1 1
"four" → 0 1 0 0
+1
+1
"five" → 0 1 0 1
+1
"six" → 0 1 1 0
"seven" → 0 1 1 1
"eight" → 1 0 0 0
+1
+1
+1

we add 1 each time

When we hit 1, we've run out of symbols, so when we add 1, we start back at 0 and increment the next place

here, when we add 1, the first 1 goes to 0 and we add one to the "twos" place. Then, the "twos" place goes to 0 and we add 1 to the "fours" place.

same sort of thing here

. . .

and so on.

Bi- means 2 in Latin

Binary is a base-2 system, which means:

- there are 2 digits (0 and 1) that can go in each position.
  - → we called a binary digit a "bit"
- each "bit" b in a binary number represents $b \cdot 2^i$
  (i is the position starting at 0 on the right)

Since binary uses symbols that decimal also uses, we sometimes run into problems

101

is this "one hundred one" in decimal?

or "five" in binary?

For clarity, we sometimes write the base along with the number.

101_{10} ← base-10, so "one hundred one"

101_2 ← base-2, so "five"

(next page)

So, how do we convert between decimal and binary?

Converting from binary to decimal

→ from right to left, add the product of the digit and 2^i where i is the position (starting with $i=0$)

Examples

①

$$11101_2 = (1\cdot 2^0)+(0\cdot 2^1)+(1\cdot 2^2)+(1\cdot 2^3)+(1\cdot 2^4)$$
$$= (1\cdot 1)+(0\cdot 2)+(1\cdot 4)+(1\cdot 8)+(1\cdot 16)$$
$$= 1+0+4+8+16$$
$$= 29_{10}$$

1	1	1	0	1
2^4	2^3	2^2	2^1	2^0

②

$$001010_2 = (0\cdot 2^0)+(1\cdot 2^1)+(0\cdot 2^2)+(1\cdot 2^3)+(0\cdot 2^4)+(0\cdot 2^5)$$
$$=(0\cdot 1)+(1\cdot 2)+(0\cdot 4)+(1\cdot 8)+(0\cdot 16)+(0\cdot 32)$$
$$= 0+2+0+8+0+0$$
$$= 10_{10}$$

0	0	1	0	1	0
2^5	2^4	2^3	2^2	2^1	2^0

Try converting 1010_{10} to binary.

Then, try converting 10111_{10} to binary

This is helpful to know before the next page

A quick aside on remainder division

Remainder division is like long division but without calculating the decimal places

For example:

$$\begin{array}{r} 1\;\; R2 \\ 3\overline{)\;5} \\ -3 \\ \hline 2 \end{array}$$

This is saying 3 can go into 5 exactly one time and then you're left with 2

$$\begin{array}{r} 6\;\; R3 \\ 4\overline{)\,27} \\ -24 \\ \hline 3 \end{array}$$

4 goes into 27 exactly 6 times with 3 left over.

$$\begin{array}{r} 13\;\; R0 \\ 2\overline{)\,26} \\ -2\downarrow \\ \hline 06 \\ -6 \\ \hline 0 \end{array}$$

2 goes into 26 exactly 13 times with nothing (0) left over.

$$\begin{array}{r} 0\;\; R1 \\ 2\overline{)\;1} \\ -0 \\ \hline 1 \end{array}$$

2 goes into 1 exactly 0 times with 1 left over.

Converting from decimal to binary

binary is base 2.

→ repeatedly divide by 2 until the result is 0 and keep track of the remainders

→ the binary number is the remainders (from top to bottom)

Examples

① 19_{10}

the base

0 R 1
2 | 1 R 0
2 | 2 R 0
2 | 4 R 1
2 | 9 R 1
2 | 19

This is the binary number (top to bottom)

= $\underbrace{10011}_{}{}_2$

Careful : If you do division like this:

2 | 19 R 1
2 | 9 R 1
2 | 4 R 0
2 | 2 R 0
2 | 1 R 1
0

Then the binary should be read bottom-to-top

② 103_{10}

the base

0 R 1
2 | 1 R 1
2 | 3 R 0
2 | 6 R 0
2 | 12 R 1
2 | 25 R 1
2 | 51 R 1
2 | 103

= $\underbrace{110111}_{}{}_2$

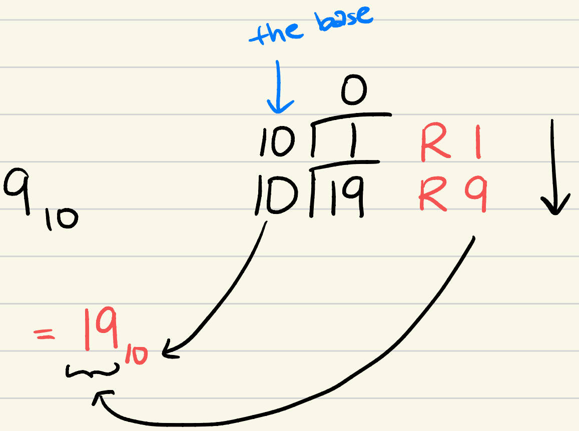
Notice that there's a remainder of 1 when the number is odd
and there's a remainder of 0 when the number is even.

Try converting 271_{10} to binary

Why does dividing by 2 like this get us the binary?

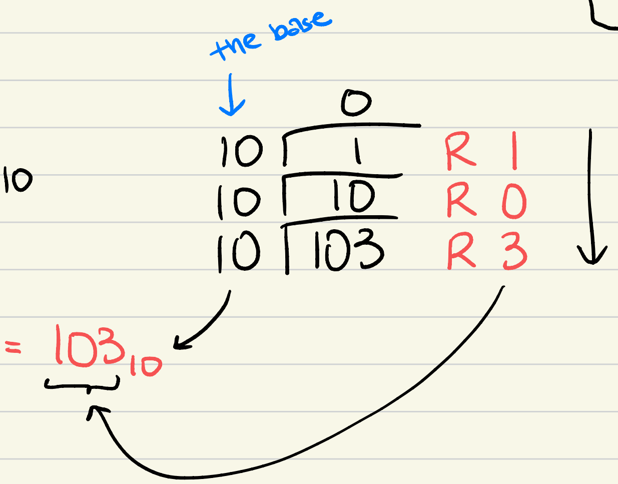
What if we tried this approach using $\overset{\text{base}}{10}$ instead of $\overset{\text{base}}{2}$?

① 19_{10}


the base
0
10 | 1 R 1
10 | 19 R 9
= 19_{10}

10 is the base here, and repeatedly dividing by 10 and keeping the remainders gets us the number in base-10.

② 103_{10}


the base
0
10 | 1 R 1
10 | 10 R 0
10 | 103 R 3
= 103_{10}

Top-to-bottom

So, if we want a base-2 number, we repeatedly divide by 2 and keep the remainders.

So, what we've learned about binary and decimal can be applied to any number system with any base.

For example:

Just some fun Examples
- base-3 is called ternary (symbols: 0, 1, and 2)
- base-4 is called quaternary (symbols: 0, 1, 2, and 3)

For the purposes of computer logic / digital design, we care about the base-16 number system (in addition to base-10 and base-2)

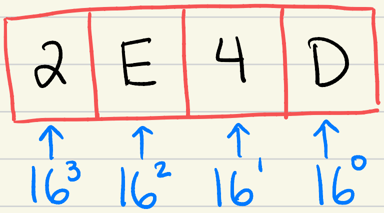
The base-16 number system is called hexadecimal.

(next page)

Hexadecimal

- Hexadecimal is base-16, so that means each position represents a power of 16, and we have 16 unique symbols that can go in each position.
- since we don't have 16 unique digits (we just have ten: 0 through 9), we need to come up with 6 more to represent 10 through 15.
- The people who came up with this stuff years ago decided to use the first six letters of the alphabet to represent 10 through 15.
- So, this is what that looks like:

Decimal Number being represented	Unique Hexadecimal Symbol
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F



You interpret this the same as binary or decimal, its just a different base.

$$(D \cdot 16^0) + (4 \cdot 16^1) + (E \cdot 16^2) + (2 \cdot 16^3)$$

Once we have this, we can substitute in the decimal numbers for the letters A through F.

$$(13 \cdot 16^0) + (4 \cdot 16^1) + (14 \cdot 16^2) + (2 \cdot 16^3)$$

Expand the exponents:

$$(13 \cdot 1) + (4 \cdot 16) + (14 \cdot 256) + (2 \cdot 4{,}096)$$

$$13 + 64 + 3{,}584 + 8{,}192$$

$$= 11{,}853_{10}$$

So $2E4D_{16}$ in hexadecimal iss the same as 11853_{10} in decimal

We use hexadecimal in computer science as a shorthand for binary. So it's important to understand how to convert between binary and hexadecimal

Converting Hexadecimal to Binary

To convert a hexadecimal number into binary,
we convert each digit to decimal
and then convert each decimal number to 4-bit binary

Example

we often refer to hexadecimal as just "hex

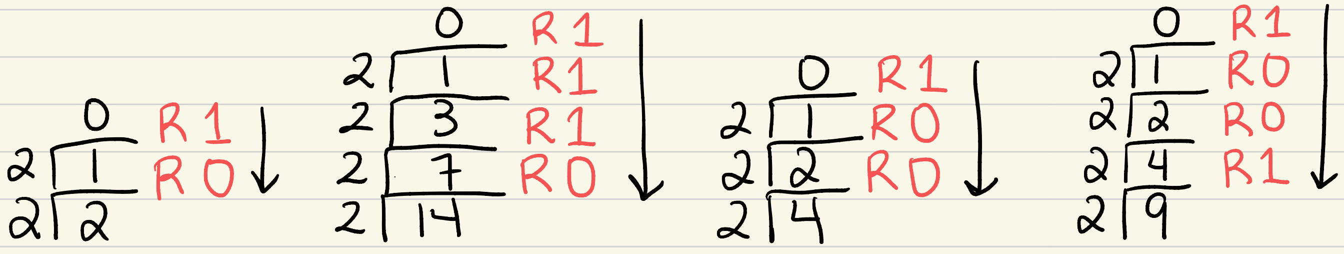
$2E49_{16}$

Hex	Decimal	Binary	4-bit Binary
2	2	10	0010
E	14	1110	1110
4	4	100	0100
9	9	1001	1001

① Write decimal

② Then, convert these to binary

③ add 0s on the left to make them all 4-bits long



$2_{10} = 10_2$ $14_{10} = 1110_2$ $4_{10} = 100_2$ $9_{10} = 1001_2$

④ Write the 4-bit binary numbers in the same order as hex

$\boxed{\text{So}: 2E49_{16} = 0010\ 1110\ 0100\ 1001_2}$ ← Solution

Try converting ABC_{16} to binary

Try converting $F9CD_{16}$ to binary

Converting Binary to Hexadecimal

To convert a binary number to hexadecimal, section the binary number into 4-bit chunks then convert each chunk to decimal, then hexadecimal

Example

$$\underbrace{0110}\,\underbrace{1001}\,\underbrace{1011}_2$$

4-bit chunks

Binary	Decimal	Hex
0110	6	6
1001	9	9
1011	11	B

① convert to decimal

convert to hex ②

$$\begin{aligned} 0110_2 &= (0\cdot 2^0)+(1\cdot 2^1)+(1\cdot 2^2)+(0\cdot 2^3) \\ &= 0+2+4+0 \\ &= 6_{10} \end{aligned}$$

$$\begin{aligned} 1001_2 &= (1\cdot 2^0)+(0\cdot 2^1)+(0\cdot 2^2)+(1\cdot 2^3) \\ &= 1+0+0+8 \\ &= 9_{10} \end{aligned}$$

$$\begin{aligned} 1011_2 &= (1\cdot 2^0)+(1\cdot 2^1)+(0\cdot 2^2)+(1\cdot 2^3) \\ &= 1+2+0+8 \\ &= 11_{10} \end{aligned}$$

③ Write the hex in order

$$\boxed{011010011011_2 = 69B_{16}}$$

↑ Solution

If you can't evenly divide into 4 bit chunks, add 0s on the left until you can.

Example

$$\underbrace{0010}\,\underbrace{1101}{}_2$$

4-bit chunks

Binary	Decimal	Hex
0010	2	2
1101	13	D

So, $101101_2 = 2D_{16}$

Solution

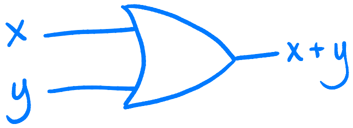
Try converting 011011110101_2 to hex

Try converting 101011100_2 to hex

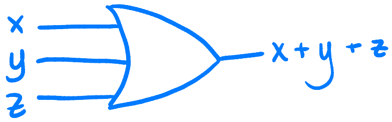
LOGIC GATES & CIRCUITS

① Draw the gates for each of the following Boolean expressions

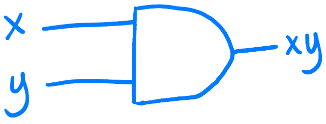
Ⓐ $x + y$



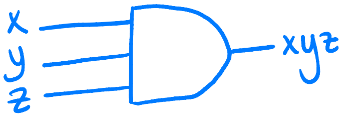
Ⓑ $x + y + z$



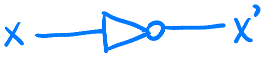
Ⓒ xy



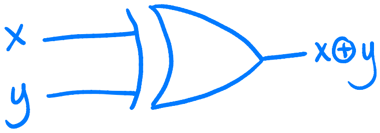
Ⓓ xyz



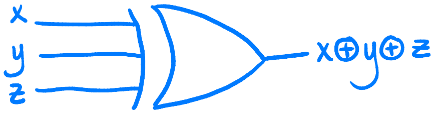
Ⓔ x'



Ⓕ $x \oplus y$

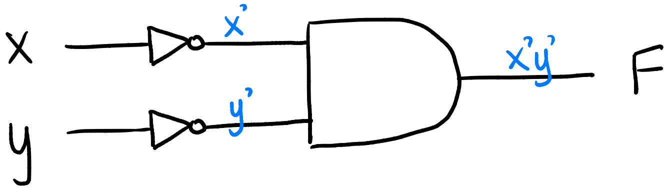


Ⓖ $x \oplus y \oplus z$



② Derive the Boolean Formula and truth table for each of the following circuits:

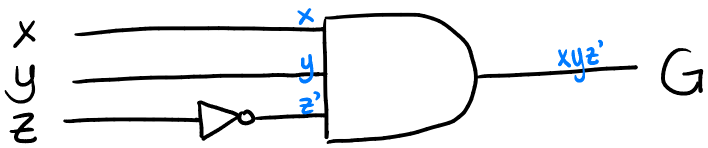
Ⓐ



x	y	x'y'
0	0	1
0	1	0
1	0	0
1	1	0

$F(x,y) = x'y'$

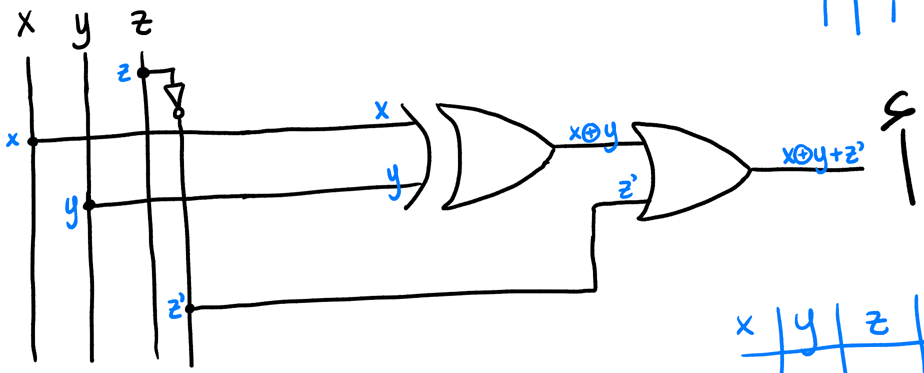
Ⓑ


x
y
z
x
y
z'
xyz'
G

x	y	z	xyz'
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

$G(x,y,z) = xyz'$

Ⓒ

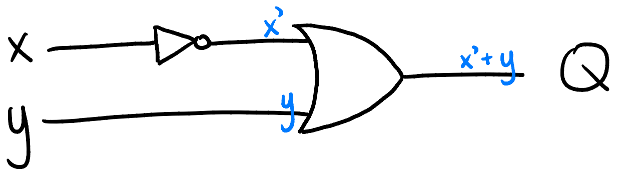

x
y
z
z
x
y
z'
x
y
x⊕y
z'
x⊕y+z'
Ϛ

x	y	z	x⊕y+z'
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

$Ϛ(x,y,z) = x \oplus y + z'$

② continued

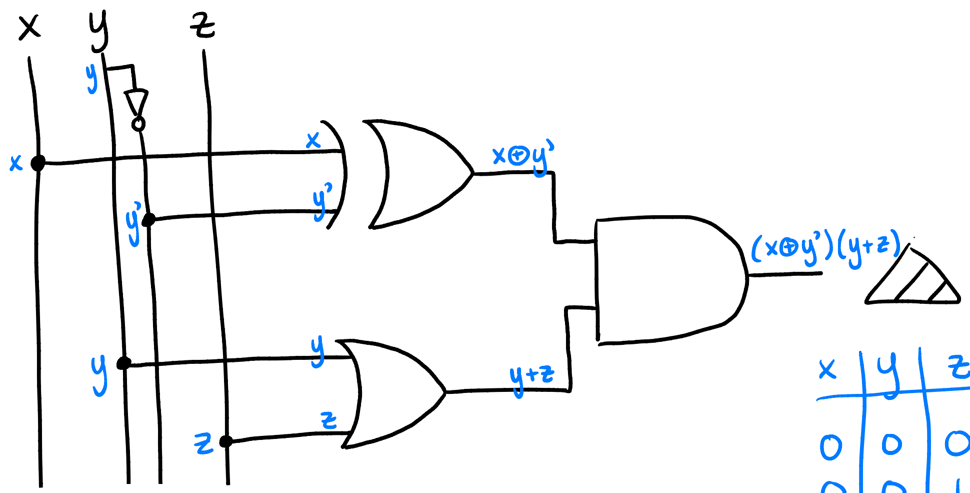
Ⓓ



$Q(x,y) = x'+y$

x	y	Q
0	0	1
0	1	1
1	0	0
1	1	1

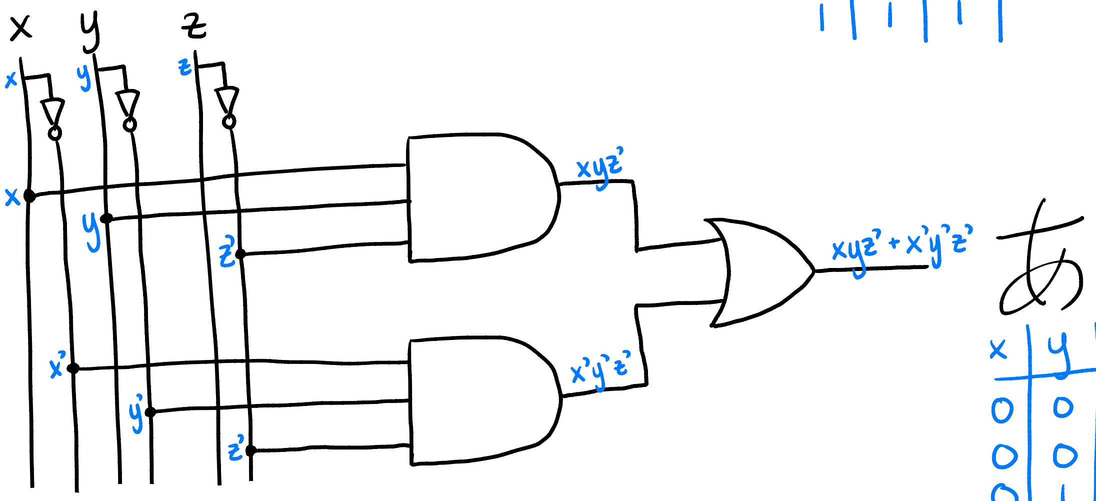
Ⓔ


x
y
z
y
x
x
y'
y'
x⊕y'
y
y
z
z
y+z
(x⊕y')(y+z)
△

$△(x,y,z) = (x\oplus y')(y+z)$

x	y	z	△
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Ⓕ

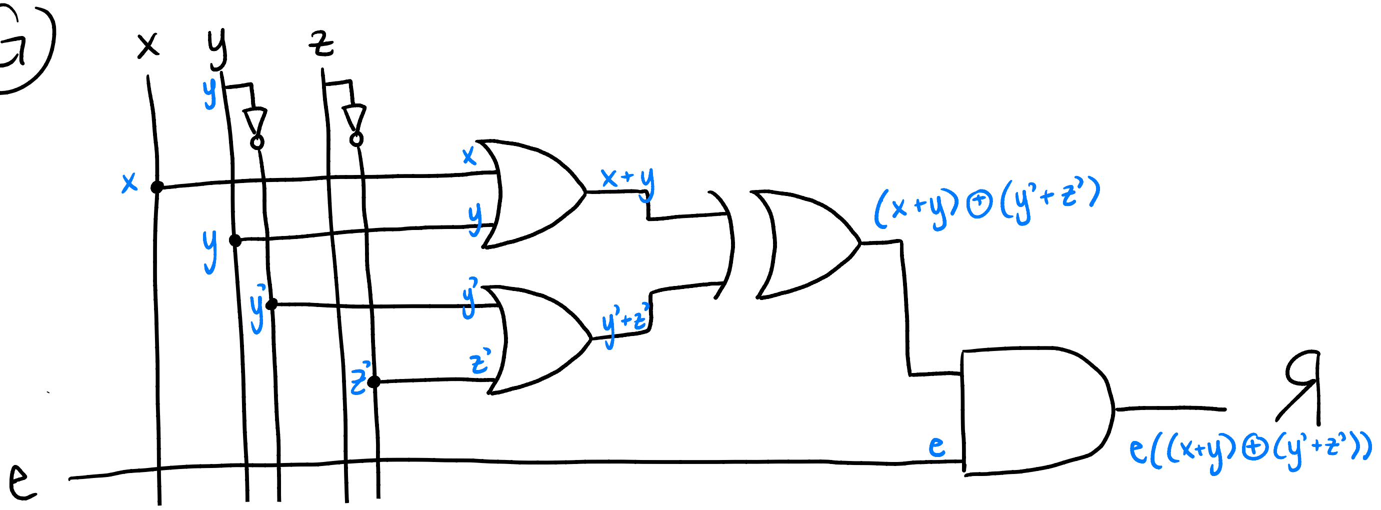

x
y
z
x
y
z
x
y
z'
xyz'
x'
y'
z'
x'y'z'
xyz'+x'y'z'
あ

あ$(x,y,z) = xyz' + x'y'z'$

x	y	z	あ
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

② continued

Ⓖ

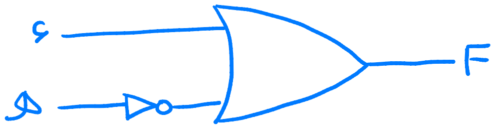

x
y
z
y
x
y
y'
z'
e
x
y
y'
z'
x+y
y'+z'
(x+y)⊕(y'+z')
e
Я
e((x+y)⊕(y'+z'))

$$\text{Я}(e,x,y,z) = e((x+y)\oplus(y'+z'))$$

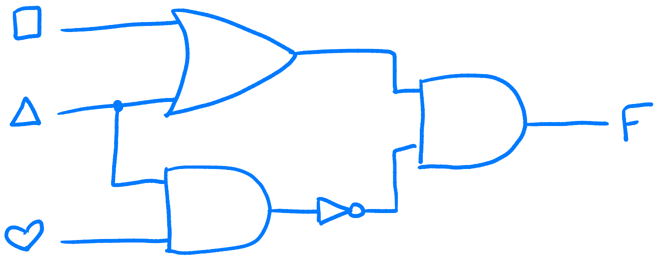
e	x	y	z	Я
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

③ Draw the circuit for each of the following Boolean Formulas :

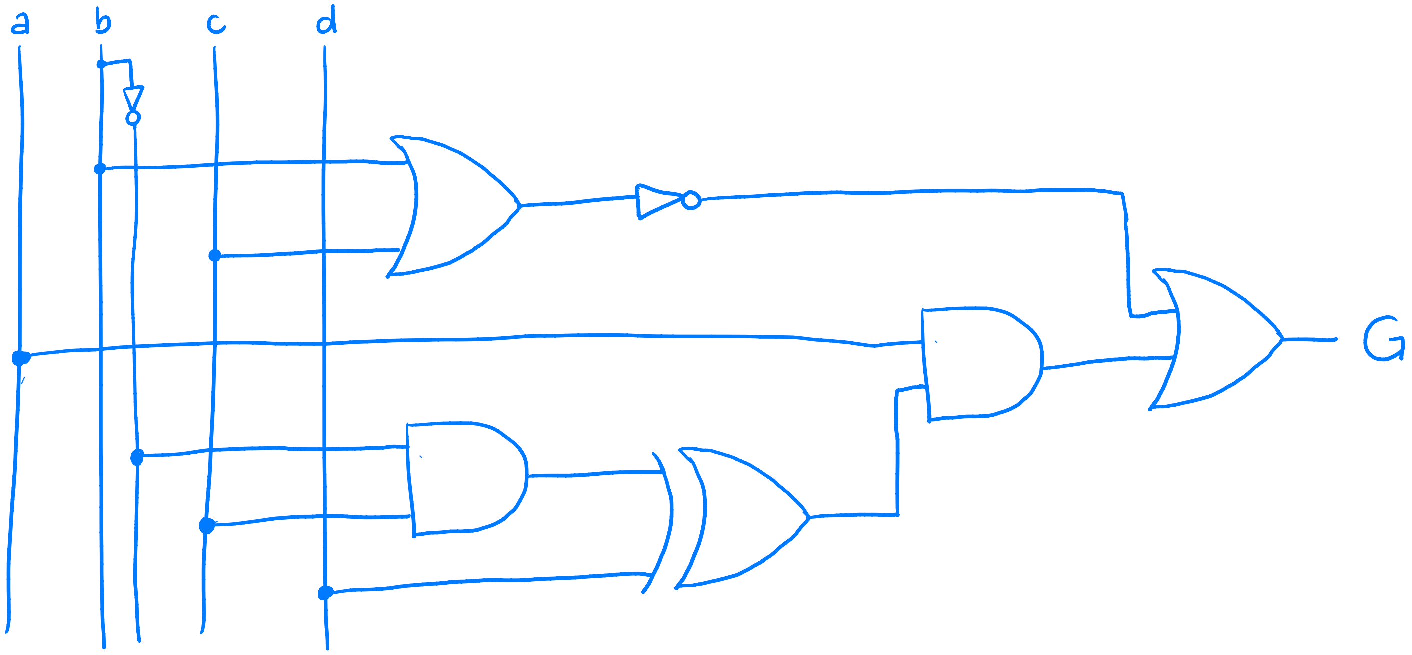
Ⓐ F(ع, ه) = ع + ه'



Ⓑ $F(\square, \triangle, \heartsuit) = (\square + \triangle)(\triangle\heartsuit)'$

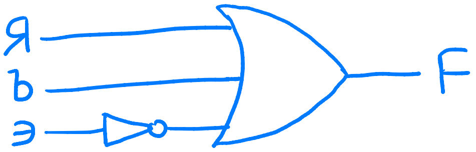


Ⓒ $G(a,b,c,d) = (b + c)' + a(b'c \oplus d)$

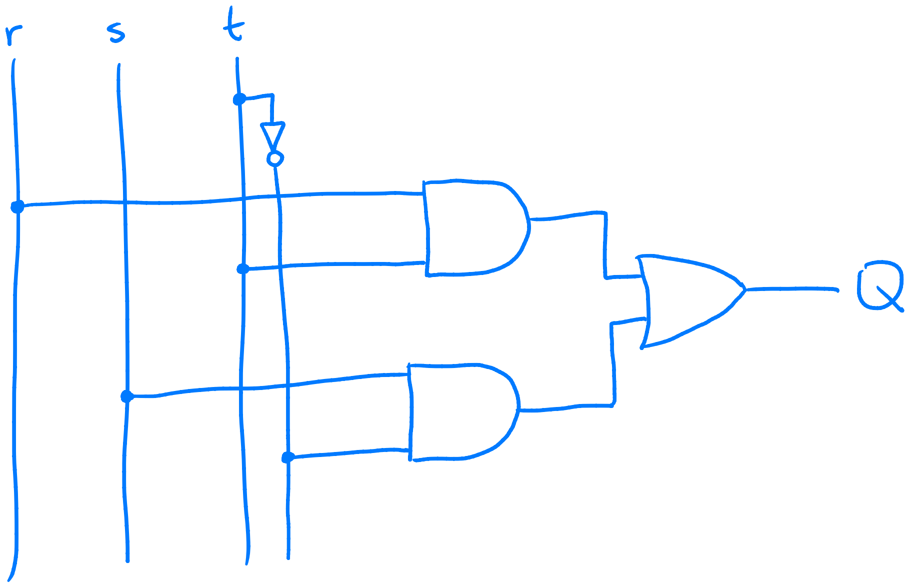


③ continued

Ⓓ $F(\text{Я}, b, \ni) = \text{Я} + b + \ni'$

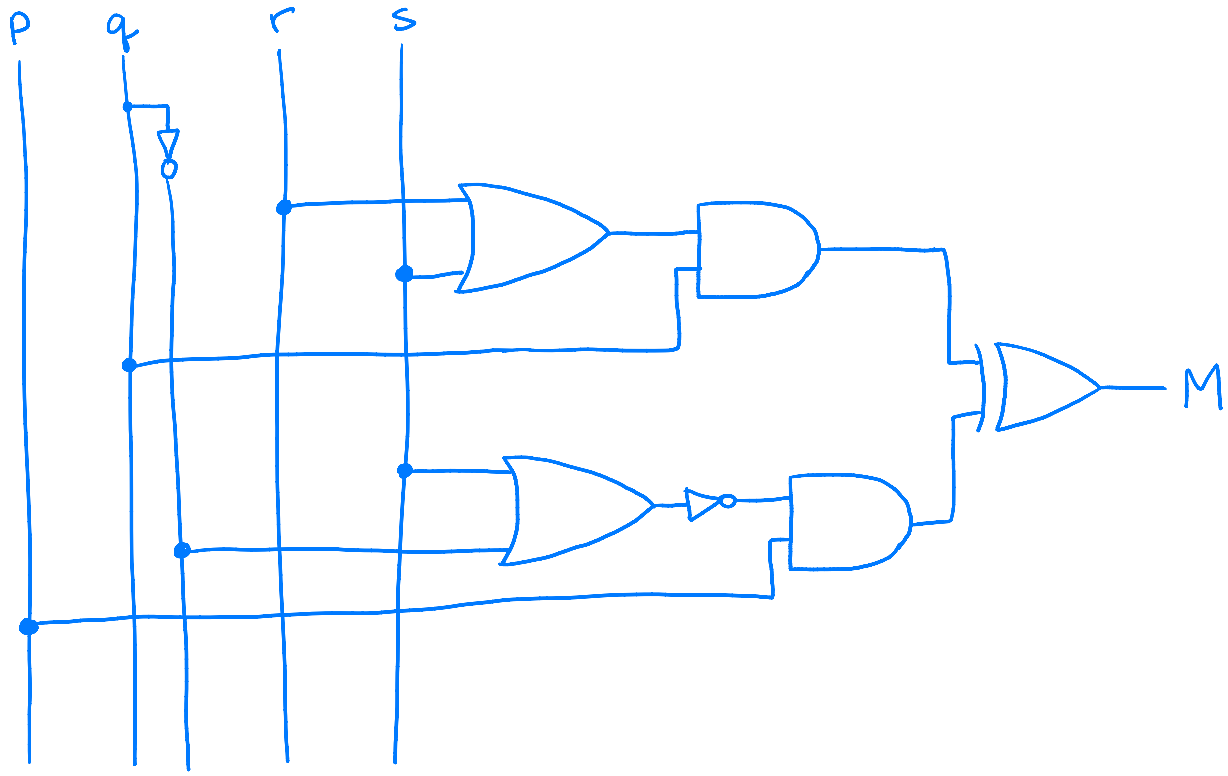


Ⓔ $Q(r, s, t) = rt + st'$



③ continued

Ⓕ $M(p,q,r,s) = q(r+s) \oplus p(s+q')'$



Circuit Diagrams

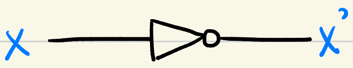
Welcome to the heart and soul of computer logic: Circuit Diagrams!

A circuit diagram is a visual representation of a Boolean function.

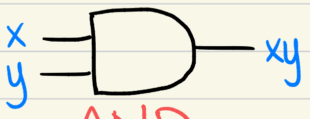
The basic building block of a circuit diagram is the **logic gate**.

Logic gates

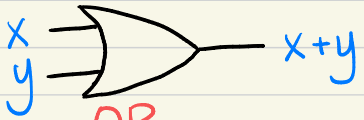
- We'll discuss 7 basic logic gates in this class.
- 4 of the 7 represent the operations we've learned so far



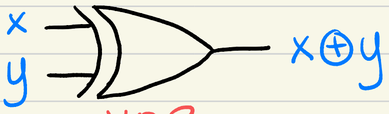
NOT



AND



OR



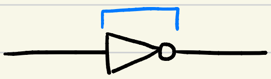
XOR

- We'll talk about the other 3 later!

Let's take a closer look at one of these logic gates to understand it's structure.

We'll use the not gate as an example. The structure of the other gates is very similar.

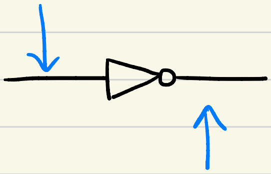
This part is the gate itself.



(The shape tells you what gate it is)

→ A triangle with a circle at the tip represents NOT

This is the input line



(sometimes called the input "signal")

This is the output line
(sometimes called the output "signal")

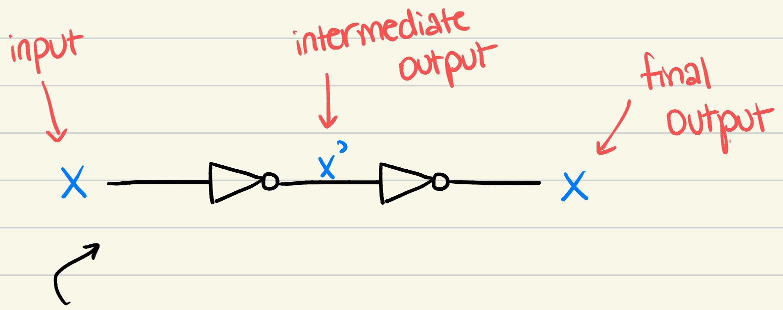
These input/output lines represent wires.

A wire can either have electricity running through it or not

(0 = electricity/on, 1 = no electricity/off)

Connecting Gates

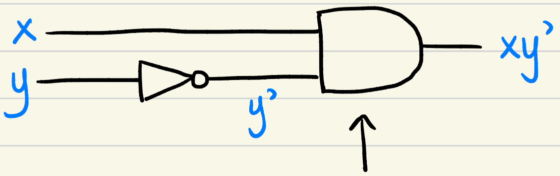
- We can connect the output of one logic gate into the input of another.


input
intermediate output
final output
x
x'
x

Note: "intermediate output" and "final output" are my wording and may not reflect how other classes describe these signals.

Here the input signal x goes through a NOT gate and becomes the output signal x'. Then x' becomes the input signal for the next NOT gate. So the final output is $(x')' = x$

Another example:

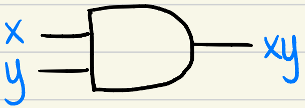


The inputs to this AND gate are x and y'

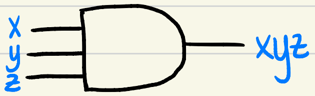
2 or more inputs

- Each gate, except for NOT, can have 2 or more inputs

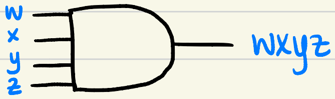
AND



2-input variant

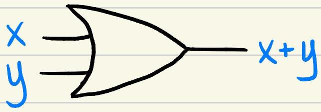


3-input variant

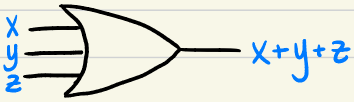


4-input variant

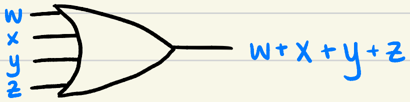
OR



2-input variant

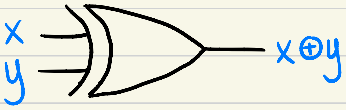


3-input variant

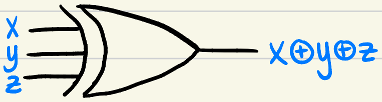


4-input variant

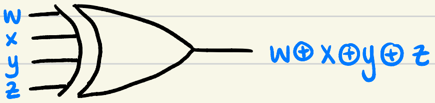
XOR



2-input variant



3-input variant



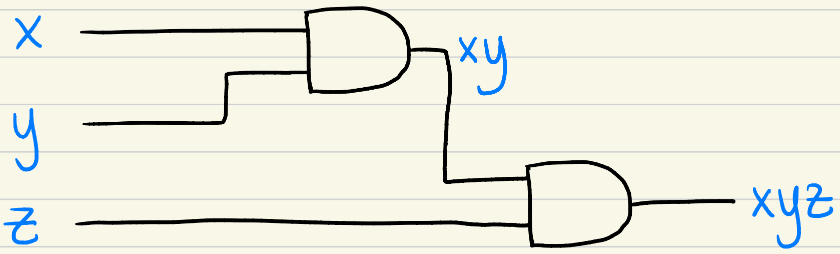
4-input variant

In the real world, there may be constraints on which version of each gate you are able to use.

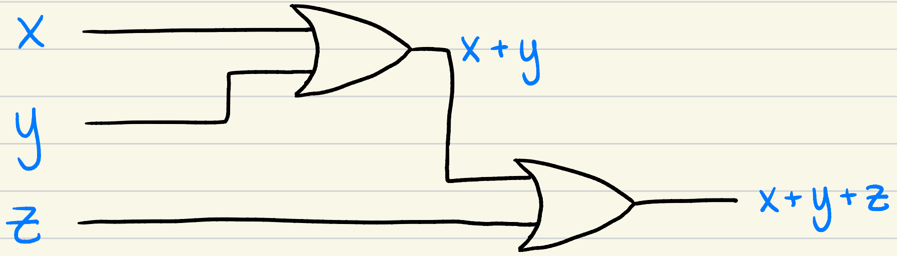
For (an oversimplified) example, you may only have access to 2-input AND gates. In this case, to perform the same function as a 3-input AND gate, you would need to use two 2-input AND gates.

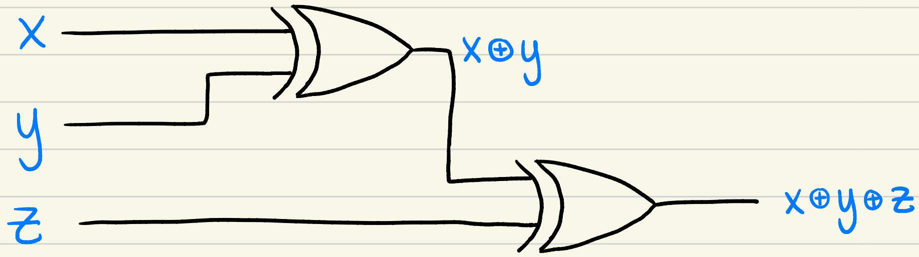
(see next page for how to do this).

3-input AND using only 2-input AND gates



The same can be done for 3-input OR and XOR





This leverages the "associativity" principle of each operation.

e.g., $xyz = (xy)z = x(yz)$

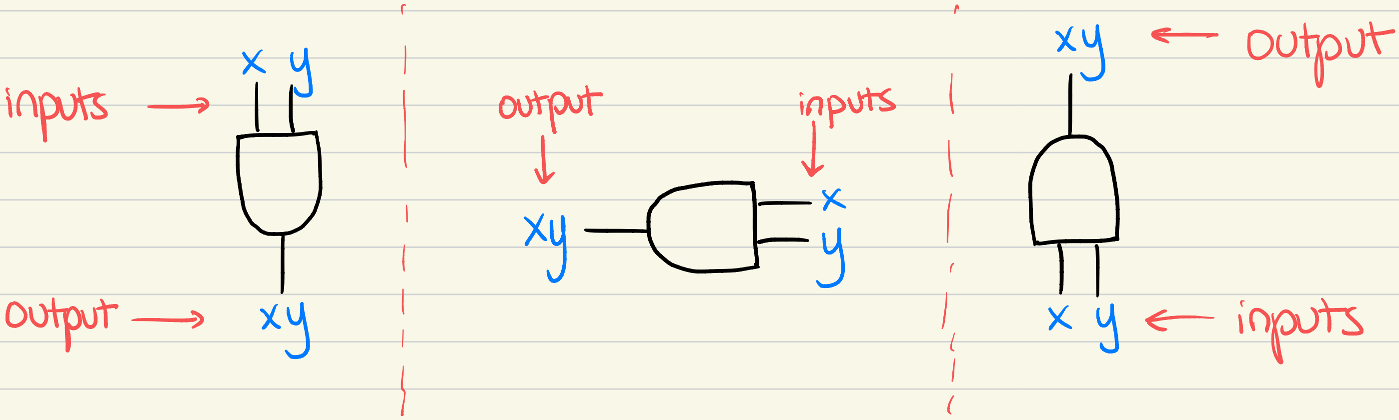
$x+y+z = (x+y)+z = x+(y+z)$

We didn't prove this but ⊕ is also commutative → $x \oplus y \oplus z = (x \oplus y) \oplus z = x \oplus (y \oplus z)$

☆ For the purposes of this class, you can use as many inputs to a single gate as necessary (NOT gates only take 1 input)

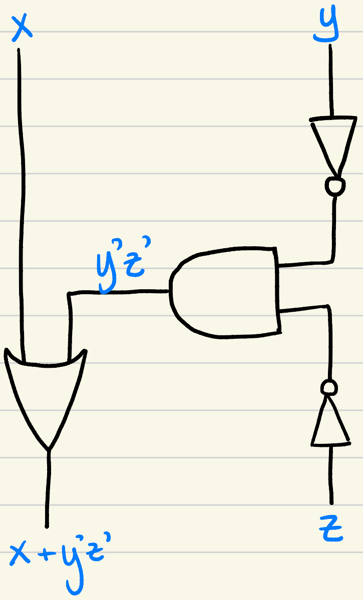
Orientation of Gates / Signals

- We could place our input and output signals "anywhere" on the diagram.
- For example, by changing the orientation of an AND gate, the position / orientation of its inputs and outputs change as well



inputs
x
y
output
xy
output
xy
inputs
x
y
xy
output
x
y
inputs

- This means that the circuit below is technically valid:



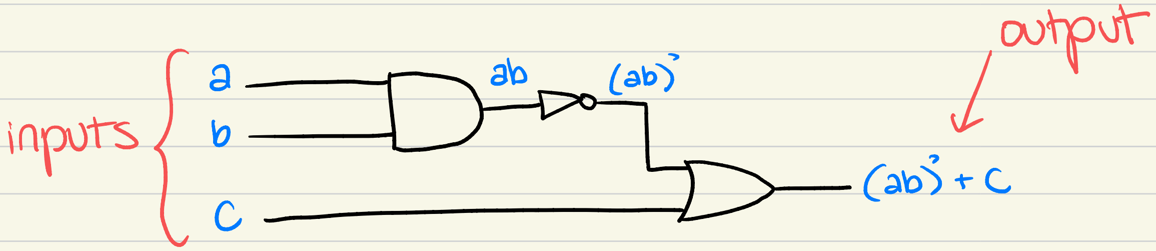
The inputs of this circuit are: x, y, z

The output is $x + y'z'$

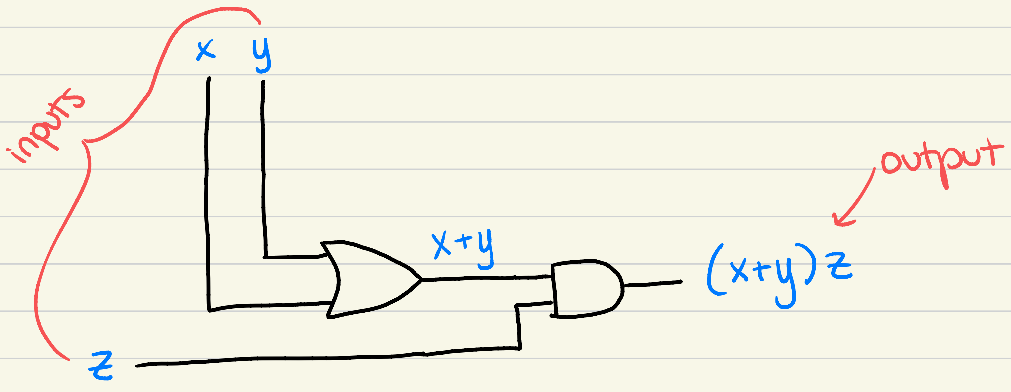
Circuit Diagram Conventions

- So that our diagrams are more organized, we typically draw our inputs at the top or on the left of the diagram.
- We typically draw our outputs on the right of the diagram.

- Examples:



output
inputs
a
b
c
ab
(ab)'
(ab)' + c



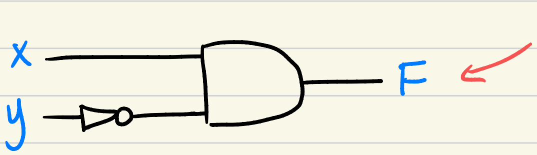
inputs
x
y
z
x+y
output
(x+y)z

☆ Note: This is not a "hard and fast" rule.

We will see this convention broken when we discuss larger black-box circuits like Adders.

Labeling outputs (Functions)

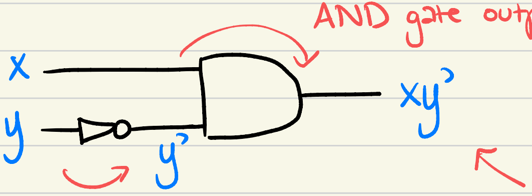
- Often, you'll see an output in a circuit diagram labeled with a single variable that is not one of the inputs
- For example:



The output here is labelled "F"

- When we see this, we can interpret that label as the name of a Boolean function determined by the circuit.

- To determine the expression for the Boolean function F, we can follow the input signals through the circuit until we get to the output.



AND gate outputs xy'

NOT gate outputs y'

The output here is xy'

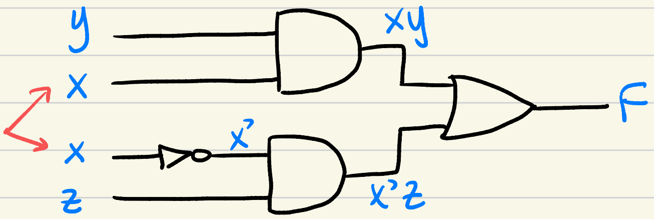
- Since the output signal labeled F is equivalent to the Boolean expression xy', we can say $F(\underbrace{x,y}_{\text{inputs}}) = xy'$

Function name

expression representing output

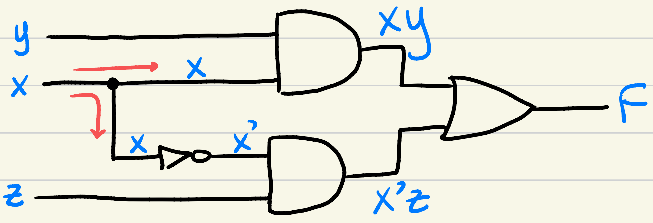
Branching wires

- Sometimes we want the same signal to go into different gates. For example, consider the circuit below for $F(x,y,z) = xy + x'z$



Your intuition might be to create two input signals with the same variable "x"

- We want to ensure that these x signals have the same value though. If electricity is running down the first x it should also be running down the second x.
- To do this, we connect these two signals by branching. The symbol we use is a filled-in dot where the branch occurs.



- Now, the single x input signal is being passed into the top AND gate and the NOT gate.

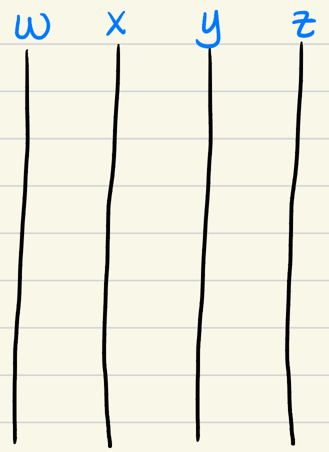
Boolean Function to Circuit Diagram (w/ "new" format)

- Now that we have branching, we can talk about a common format for designing circuit diagrams.

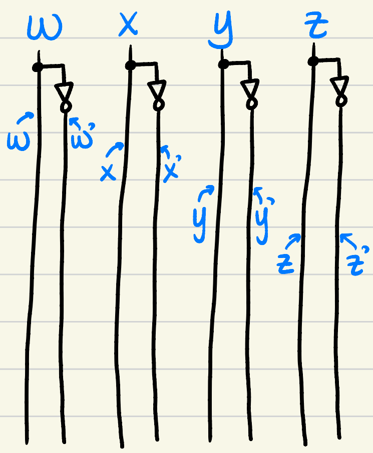
- Let's say we have the function $F(w,x,y,z) = w \oplus y + xz'$

- This function has 4 inputs: w, x, y, z

  So, we can draw these input signals as vertical lines on the left of the diagram.

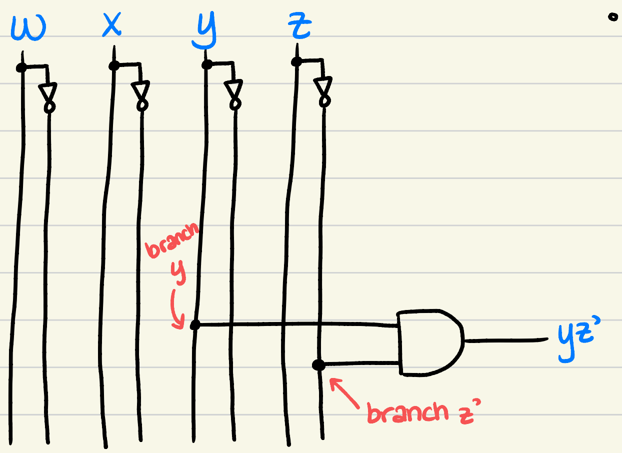


- Then, we can branch each of these signals into a NOT gate to get the signals w' x' y' z'



- The resulting signals have been labeled

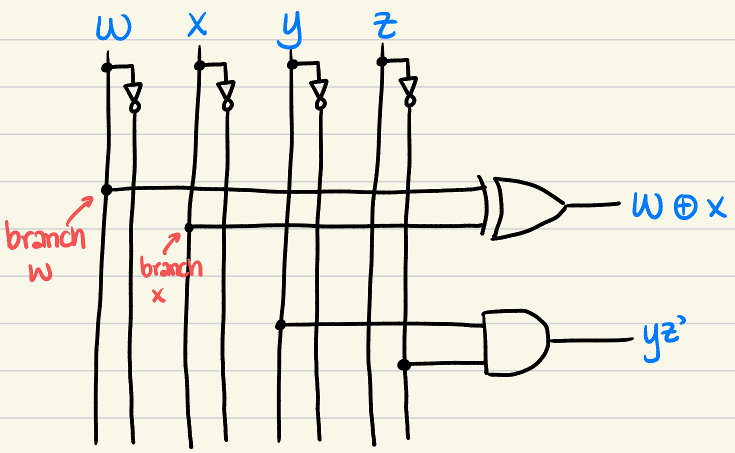
- Now, to the right of these input signals, we can draw the gates we need (using the order of operations)
- In $F(w,x,y,z) = w \oplus x + yz'$, the first gate we should take care of is the AND (we already took care of NOT)



- we want this AND gate to output yz', so its inputs should be y and z'
  - We branch y and z' from their corresponding input lines.

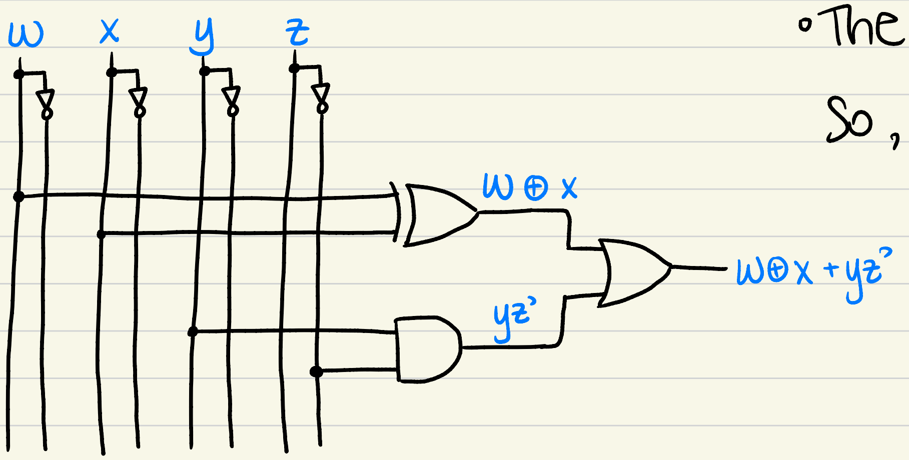
- Next, we take care of the XOR
  - The XOR is $w \oplus x$

    So, the inputs are w and x



☆ Branching only occurs when we draw a dot on the point where two lines cross (No dot, no branching)

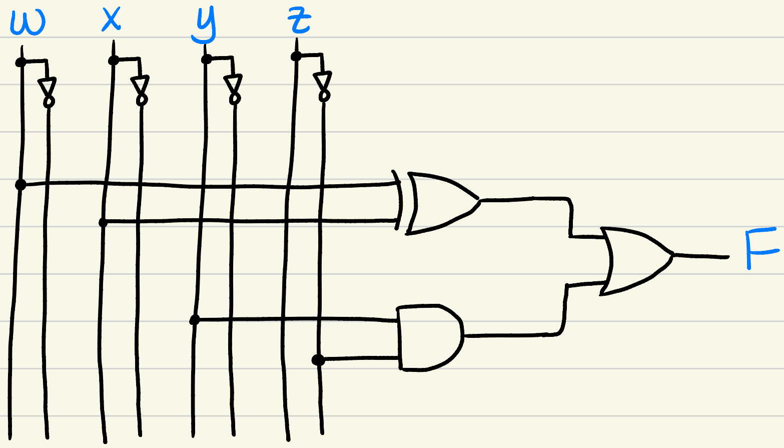
- Finally, we can take care of the OR.


w
x
y
z
$w \oplus x$
yz'
$w \oplus x + yz'$

- The OR is $\overbrace{w \oplus x} + \overbrace{yz'}$

  So, the inputs are $w \oplus x$ and yz'

- Since $F(w,x,y,z) = w \oplus x + yz'$, we can label the final output F

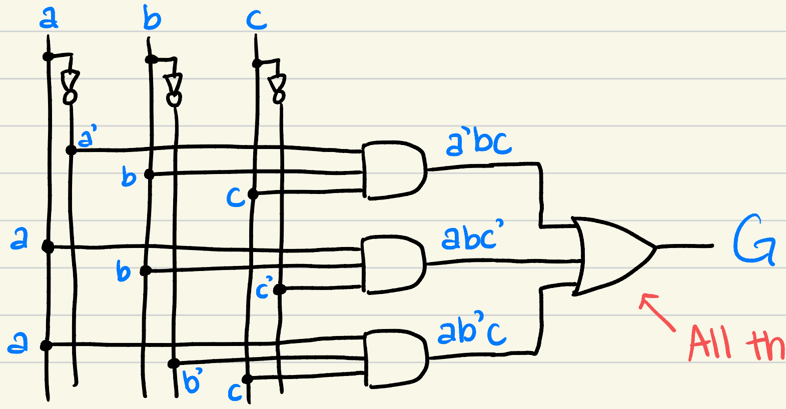


- We're done!

"Canonical" Circuit Diagrams

- Since canonical sum-of-products and canonical product-of-sums are just specific representations of Boolean functions, we can draw circuits of these representations.

- For example: $G(a,b,c) = a'bc + abc' + ab'c$ can be represented with the circuit below:



All the minterms are OR'd together

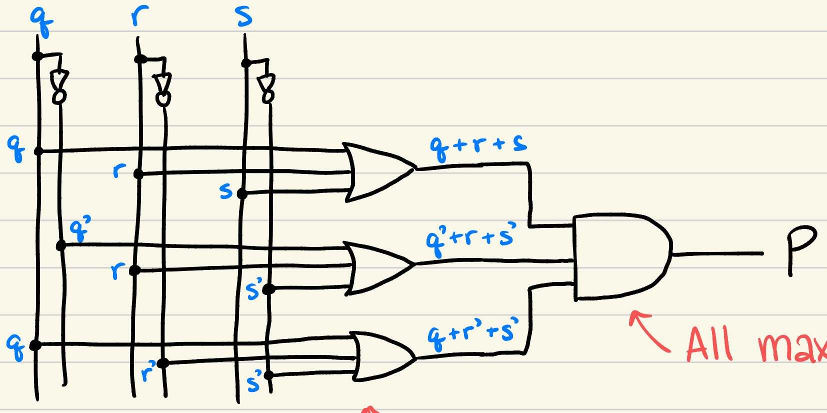
Notice how each AND gate outputs a minterm

This is a canonical sum-of-products circuit diagram.

- Another example :

$$P(q,r,s) = (q+r+s)(q'+r+s')(q+r'+s')$$

can be represented with the following circuit:


q
r
s
q
r
s
q'
r
s'
q
r'
s'
q+r+s
q'+r+s'
q+r'+s'
P

↖ All maxterms are AND'd together

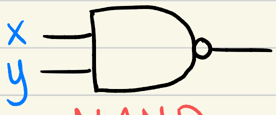
↑ Notice how each OR gate outputs a maxterm

This is a canonical product-of-sums circuit diagram.

(next page)

3 more logic gates

- So far, we've talked about 4 basic logic gates
  NOT, AND, OR, XOR
- There are 3 more basic logic gates that we'll talk about.
- These 3 represent NOT'd versions of
  AND, OR, and XOR respectively.

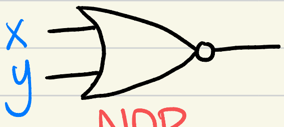


DeMorgan's

$(xy)' = x' + y'$

NAND

↑
NOT(AND)

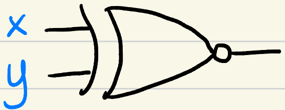


DeMorgan's

$(x+y)' = x'y'$

NOR

↑
NOT(OR)



truth table SOP

$(x \oplus y)' = x'y' + xy$

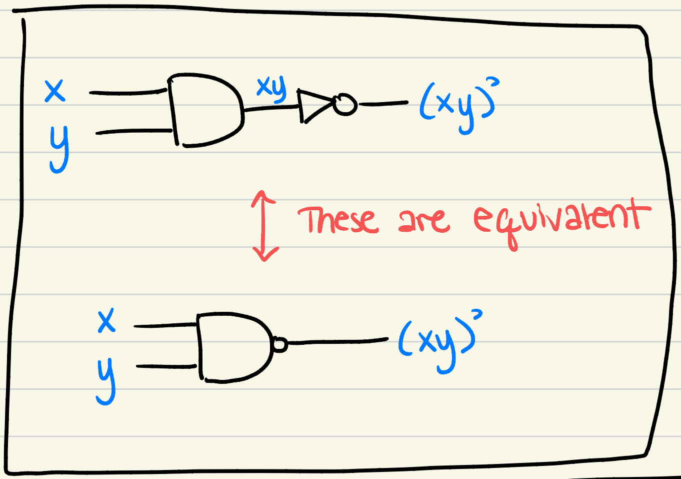
XOR	XNOR	
0	1	← $x'y'$
1	0	
1	0	
0	1	← xy

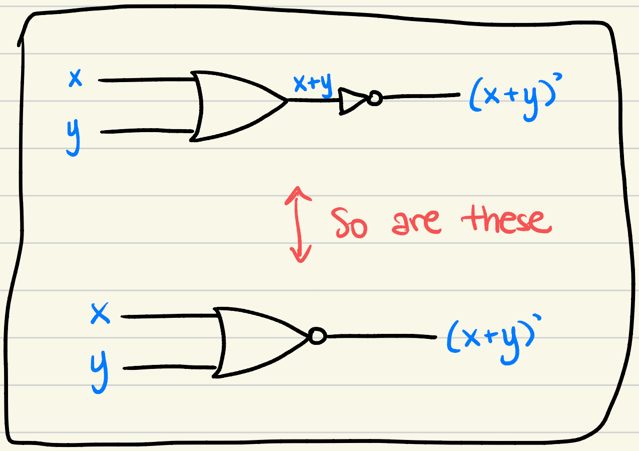
XNOR

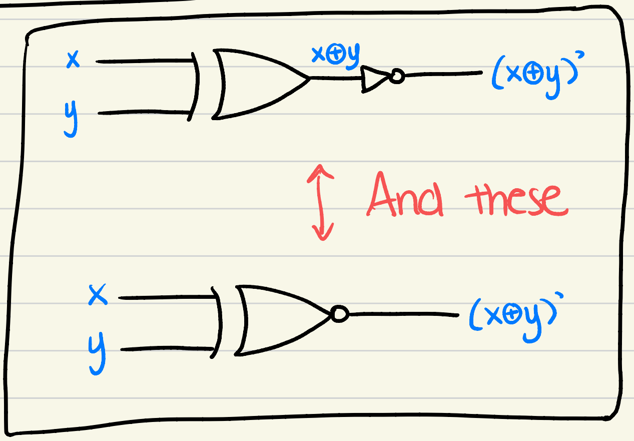
↖ NOT(XOR)

- Notice that these gates are just the gates for AND, OR, and XOR but with a circle on the output. This circle represents NOT.

These gates are like a shorthand sending the output into a NOT gate right after the corresponding gate.


x
y
xy
(xy)'
These are equivalent
x
y
(xy)'


x
y
x+y
(x+y)'
So are these
x
y
(x+y)'


x
y
x⊕y
(x⊕y)'
And these
x
y
(x⊕y)'

XNOR

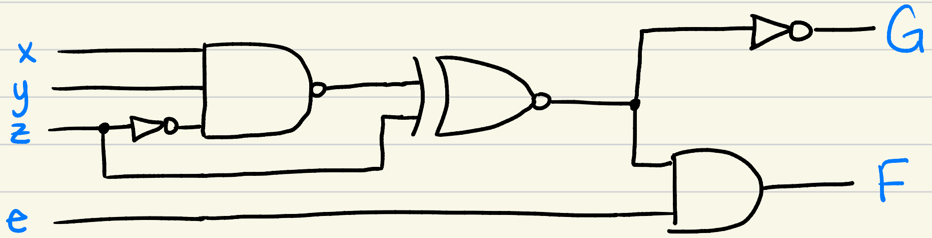
- The XNOR gate actually has its own Boolean Algebra symbol : $\odot$

$$x \odot y = (x \oplus y)' = xy + x'y'$$

- XNOR has the same priority as XOR in the order of operations [NOT → AND → XOR/XNOR → OR]
- XNOR is also commutative and associative.

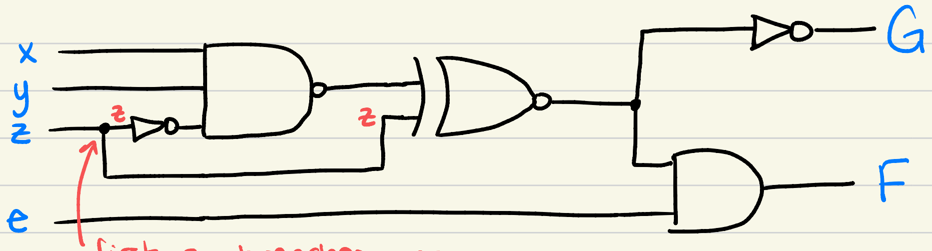
Writing Boolean expressions from a circuit diagram

- If we have a circuit diagram like the following, and we want to write a Boolean expression for each of its outputs : F, G



★ A circuit can have more than one output!

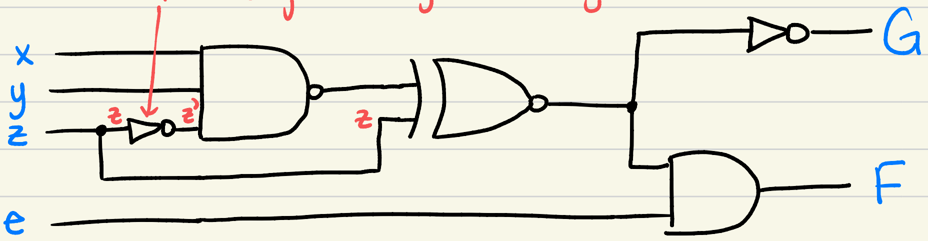
- We just need to follow the signals through the circuit.
- Start from the inputs and write the outputs of each branch / gate.

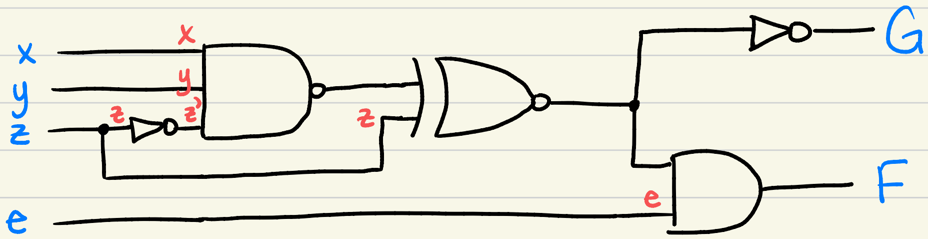


first, z branches, so z goes into the NOT gate and the XNOR.

(next page)

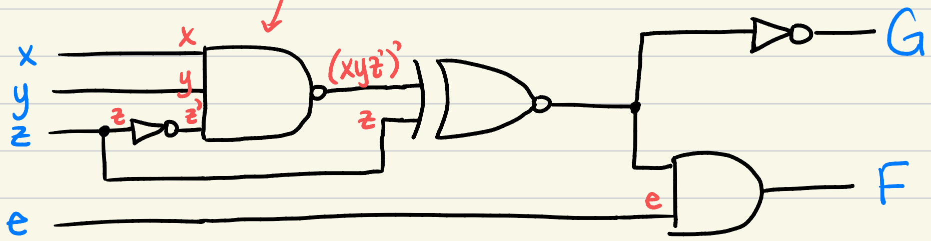
Now z goes through a NOT gate to become z'



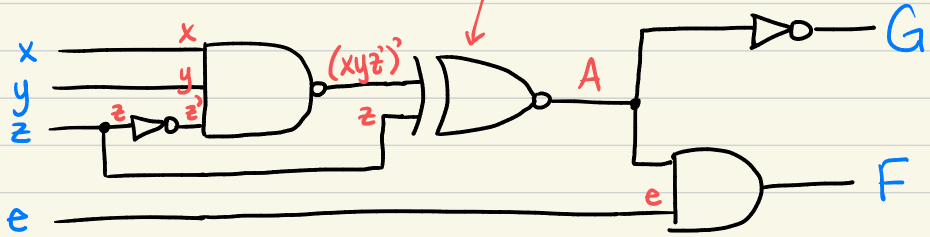

x
y
z
x
y
z
z'
z
e
e
G
F

Now, follow x, y, and e to their next gates.

x, y, and z' go into this NAND, which outputs (xyz')'


x
y
z
x
y
z
z'
(xyz')'
z
e
e
G
F

(xyz')' and z go into this XNOR, which outputs

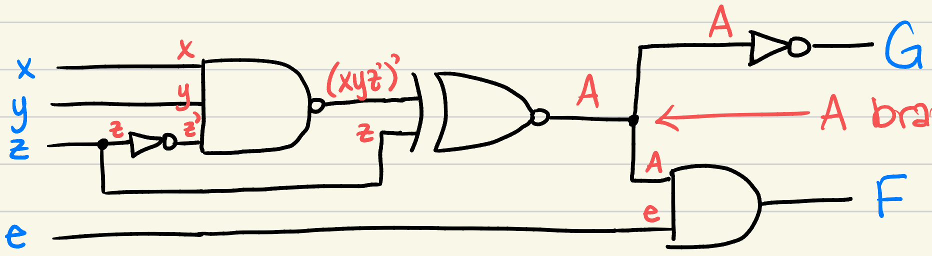

x
y
z
x
y
z
z'
(xyz')'
z
A
e
e
G
F

$(xyz')'z + (xyz')z'$

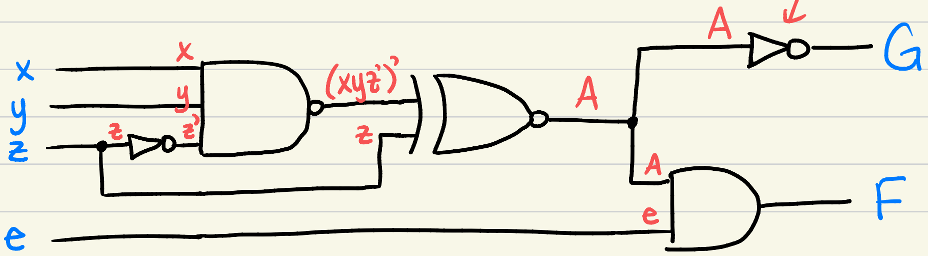
Double Complement

let's call this A

$A = (xyz')'z + (xyz')z'$


x
y
z
e
x
y
z
z'
(xyz')'
z
A
A
A
e
G
F

A branches


x
y
z
e
x
y
z
z'
(xyz')'
z
A
A
A
e
G
F

A goes through NOT gate which outputs A' (this line is labeled G)

So, $G = A'$. Remember that $A = (xyz')'z + (xyz')z'$

Then, $G = ((xyz')'z + (xyz')z')'$

$= ((x'+y'+z)z + xyz')'$

$= (x'z + y'z + z + xyz')'$

$= ((x'+y'+1)z + xyz')'$

$= (z + xyz')'$

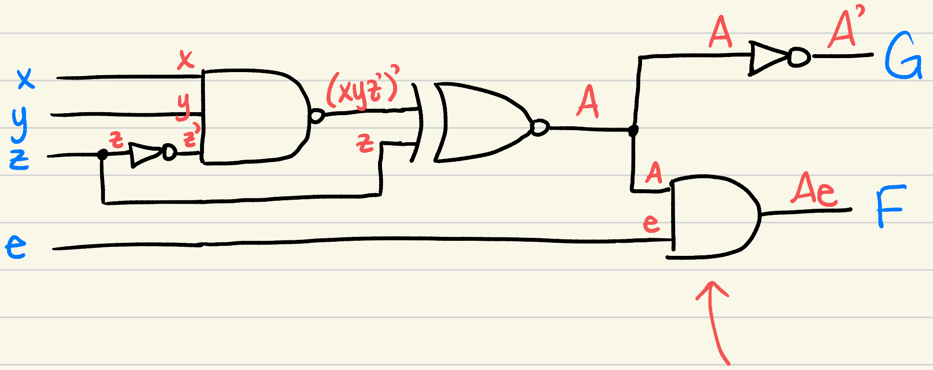
$= z'(x'+y'+z)$

$= x'z' + y'z' + \cancel{zz'}^{\to 0}$

$= x'z' + y'z'$

- Use Boolean Algebra to reduce G.

(on the next page we'll finish with F)


x
y
z
e
x
y
z
z'
z
(xyz')'
z
A
A
A'
G
A
e
Ae
F

A and e go through this AND gate, which outputs Ae.

(this output line is labeled F)

So, $F = Ae$

$= (z + xyz')e$ we reduced A on the previous page

$= ez + exyz'$

Finally, we have:

$$G(x,y,z,e) = x'z' + y'z'$$

$$F(x,y,z,e) = ez + exyz'$$

Creating Truth Tables from a circuit diagram

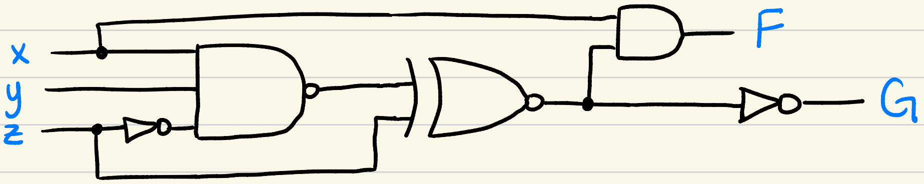
- To create the truth table for a circuit, you can:
    - Write the Boolean function for the outputs
    - Then, create the truth table for the Boolean function

  or:
    - use a "hard-wiring" technique to generate the truth table directly from the circuit.

- I'm going to focus on the hard-wiring approach here.

(see next page)

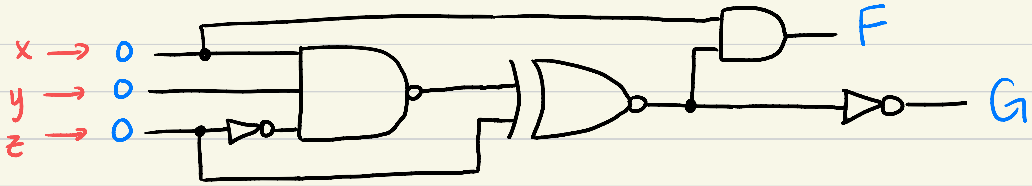
- Let's start with a simpler circuit w/ only 3 inputs



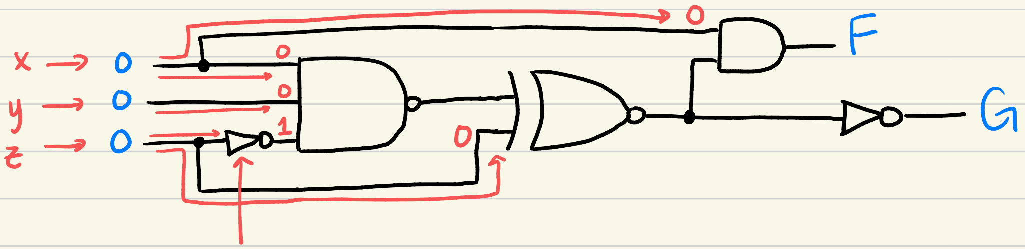
- We start by creating a truth table w/ empty output columns.

x	y	z	F	G
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

- Then for each row, we "hard-wire" the inputs of the circuit. This just means we set the input signal to either a 0 or a 1 depending on the value for that input in the current row.
- The first row of the truth table has x = 0, y = 0, z = 0.
- So we set the inputs accordingly.

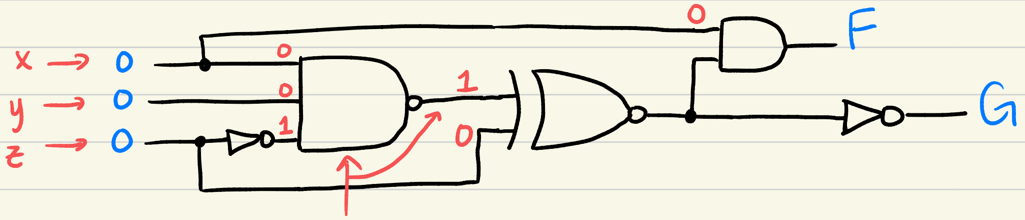


- Then we follow those inputs through the circuit


x → 0
y → 0
z → 0
0
0
0
1
0
F
G

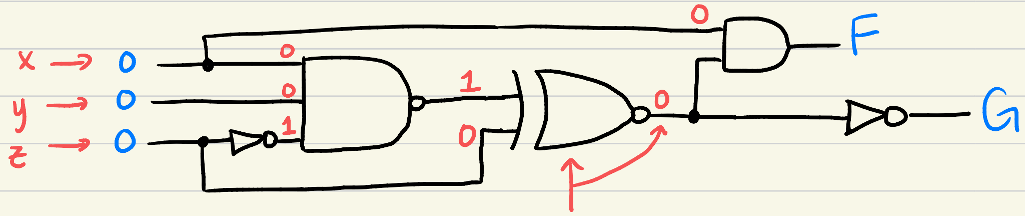
0 through a NOT becomes 1.

$(0' = 1)$


x → 0
y → 0
z → 0
0
0
0
1
1
0
F
G

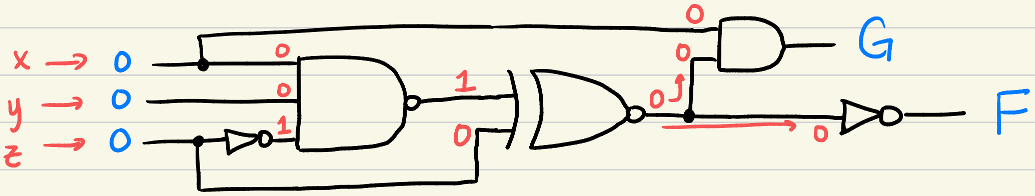
NOT (0 AND 0 AND 1)
= NOT (0)
= 1

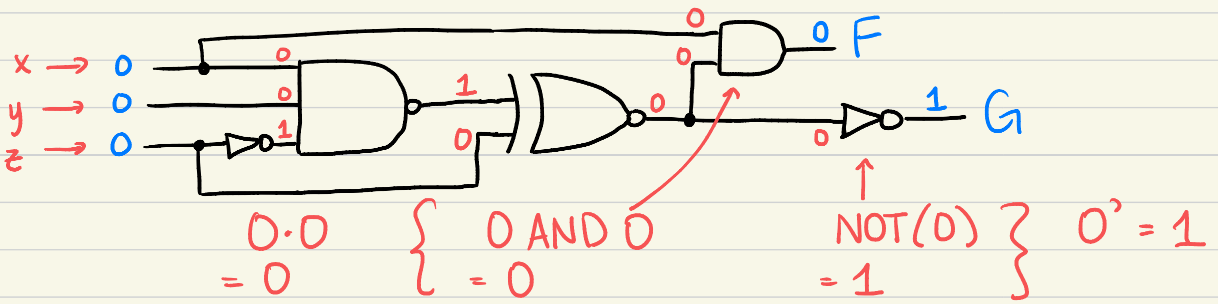
$$\left.\begin{aligned} &(0 \cdot 0 \cdot 1)' \\ &= 0' \\ &= 1 \end{aligned}\right.$$


x → 0
y → 0
z → 0
0
0
0
1
1
0
0
F
G

NOT (1 XOR 0)
= NOT (1)
= 0

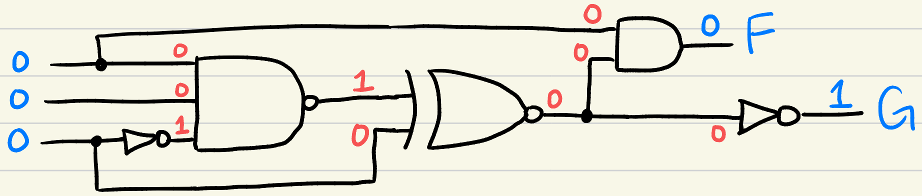
$$\left.\begin{aligned} &(1 \oplus 0)' \\ &= 1' \\ &= 0 \end{aligned}\right.$$


x → 0
y → 0
z → 0
0
0
0
0
1
1
0
0
0
G
F


x → 0
y → 0
z → 0
0·0 = 0
0 AND 0 = 0
NOT(0) = 1
0' = 1
F
G

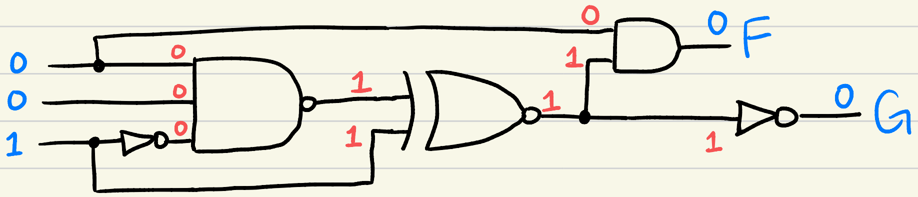
- Now that we've followed the signals through the circuit, we know that F = 0 and G = 1 when x = 0, y = 0, and z = 0.
- So we can fill out this row of the truth table

x	y	z	F	G
0	0	0	0	1
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

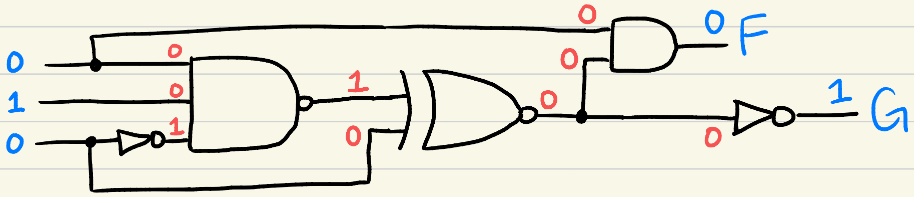


- And we can do the same for the rest of the rows.

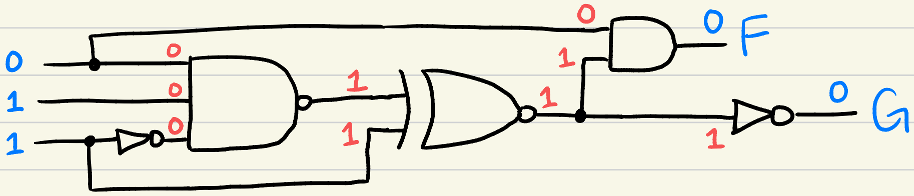
x	y	z	F	G
0	0	0	0	1
0	0	1	0	0
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		



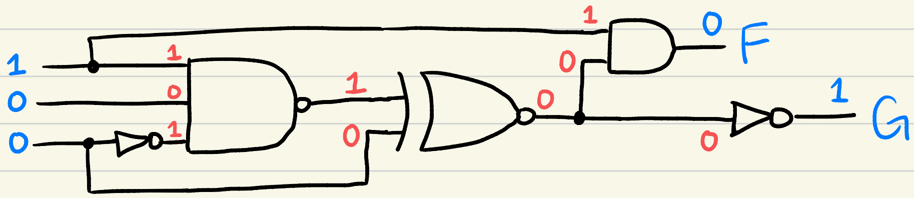
x	y	z	F	G
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		


0
0
F
0
0
0
0
1
0
1
1
0
1
G
1
0
0
0

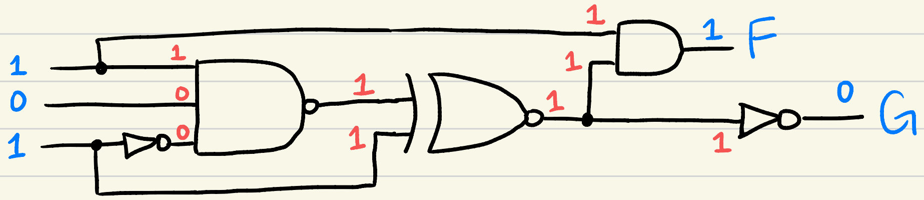
x	y	z	F	G
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	0	0
1	0	0		
1	0	1		
1	1	0		
1	1	1		


0
0
F
0
1
0
1
0
0
1
1
G
0
1
1
1

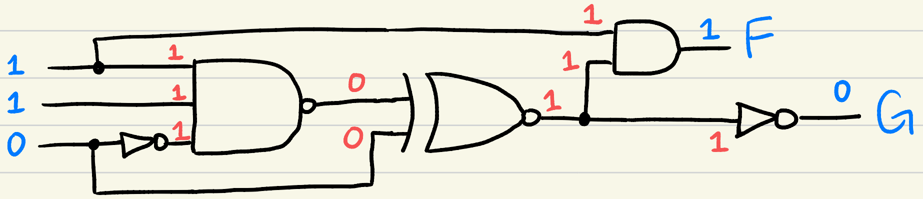
x	y	z	F	G
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	0	0
1	0	0	0	1
1	0	1		
1	1	0		
1	1	1		


1
0
F
1
1
0
1
0
1
0
0
G
1
0
0
0

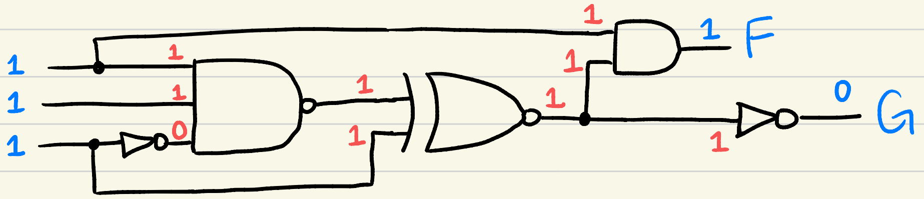
x	y	z	F	G
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0		
1	1	1		


1
1
F
1
1
1
1
0
1
0
0
G
0
1
1
1

x	y	z	F	G
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1		


1
1
0
1
1
1
1
0
0
1
1
1
1
F
1
0
G

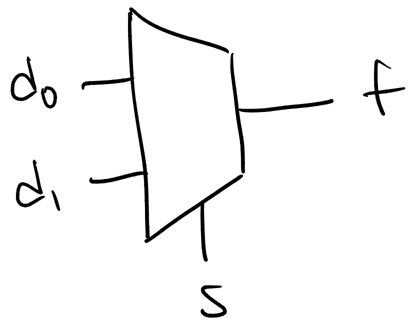
x	y	z	F	G
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0


1
1
1
1
1
0
1
1
1
1
1
1
F
1
0
G

x	y	z	F	G
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

Et voila! We have the truth table for the circuit

★ Double check that you get the same truth table from the Boolean equations.



this is a 2-to-1 multiplexer (MUX)

It has $n=1$ control line called "s".

It also has $2^n = 2^1 = 2$ data lines (d_0 and d_1)

It has one output line called "f".

The control line determines which of the data lines to pass through to f.

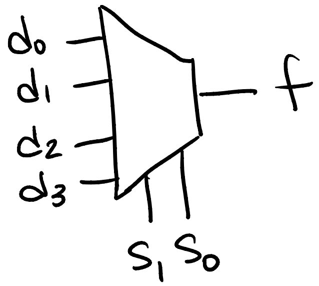
If $s=0$, then $f=d_0$

If $s=1$, then $f=d_1$

In general, we can draw any 2^n-to-1 multiplexer, where n is the number of control signals, 2^n is the number of data signals, and there is always 1 output signal.

For example:

This is a 4-to-1 MUX.



If $s = \{s_1, s_0\} = 00$, then $f = d_0$.

If $s = \{s_1, s_0\} = 01$, then $f = d_1$.

If $s = \{s_1, s_0\} = 10$, then $f = d_2$.

If $s = \{s_1, s_0\} = 11$, then $f = d_3$.

The binary number s formed by the n control bits determine which data line is passed through.

For example, in an 8-to-1 MUX,

$s = \{s_2, s_1, s_0\} = \underline{\underline{101}}$ would pass $\underline{\underline{d_5}}$ to f.

---

① Draw the circuit for each of the following types of multiplexers using just the basic logic gates.

Ⓐ 2-to-1 MUX

Ⓑ 4-to-1 MUX

Ⓒ 8-to-1 MUX

② Hardwire the data lines of a 4-to-1 MUX to represent each of the following Boolean functions (use a "black box" MUX)

Ⓐ $f(x,y) = x \oplus y'$

Ⓑ $g(p,q) = p + p'q$

② Hardwire the inputs of an 8-to-1 MUX to represent each of the following Boolean functions:
(use a "black-box" 8-to-1 MUX)

Ⓐ $f(a,b,c) = a'b + c(a \odot b)$

Ⓑ $H(v,w,x) = vw(x \odot w) + v'x(x \oplus w)$

③ Using only 2-to-1 MUXs, build a circuit that encodes each of the following equations:

Ⓐ $f(p,q) = p + q$

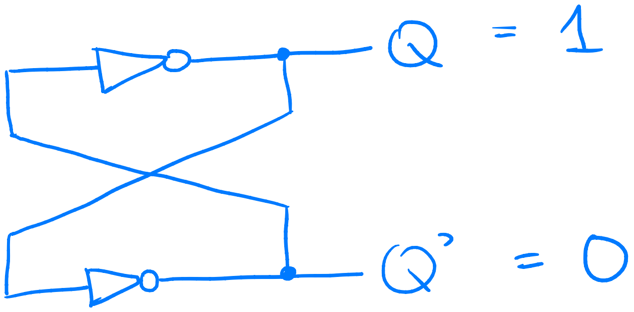
Ⓑ $N(A_1, A_2) = A_1 \oplus A_2$

Ⓒ $Q(f, e, g) = fe' + g \oplus f'$

Ⓓ $A(b, c, d) = bc + b'd'$

Latches Guided Notes

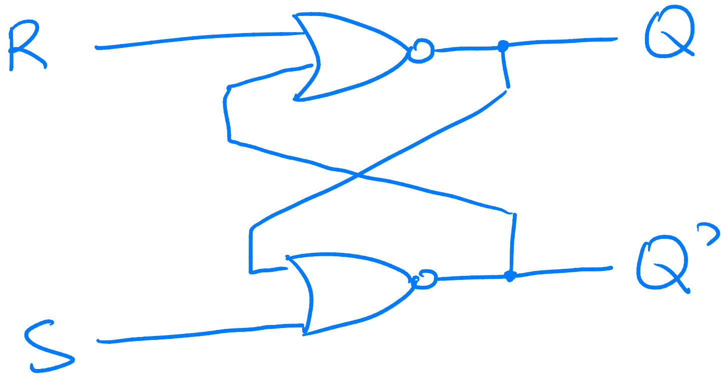
① Draw a bistable element using two NOT gates. Label the outputs Q and Q'. Then, assume the top NOT gate (with output Q) has a slightly shorter delay. What values will Q and Q' have when this circuit stabilizes?



② SR latch

Ⓐ Draw an SR latch with NOR gates.
Label the inputs R and S.
Label the outputs Q and Q'.

Then, fill out the truth table for this latch.



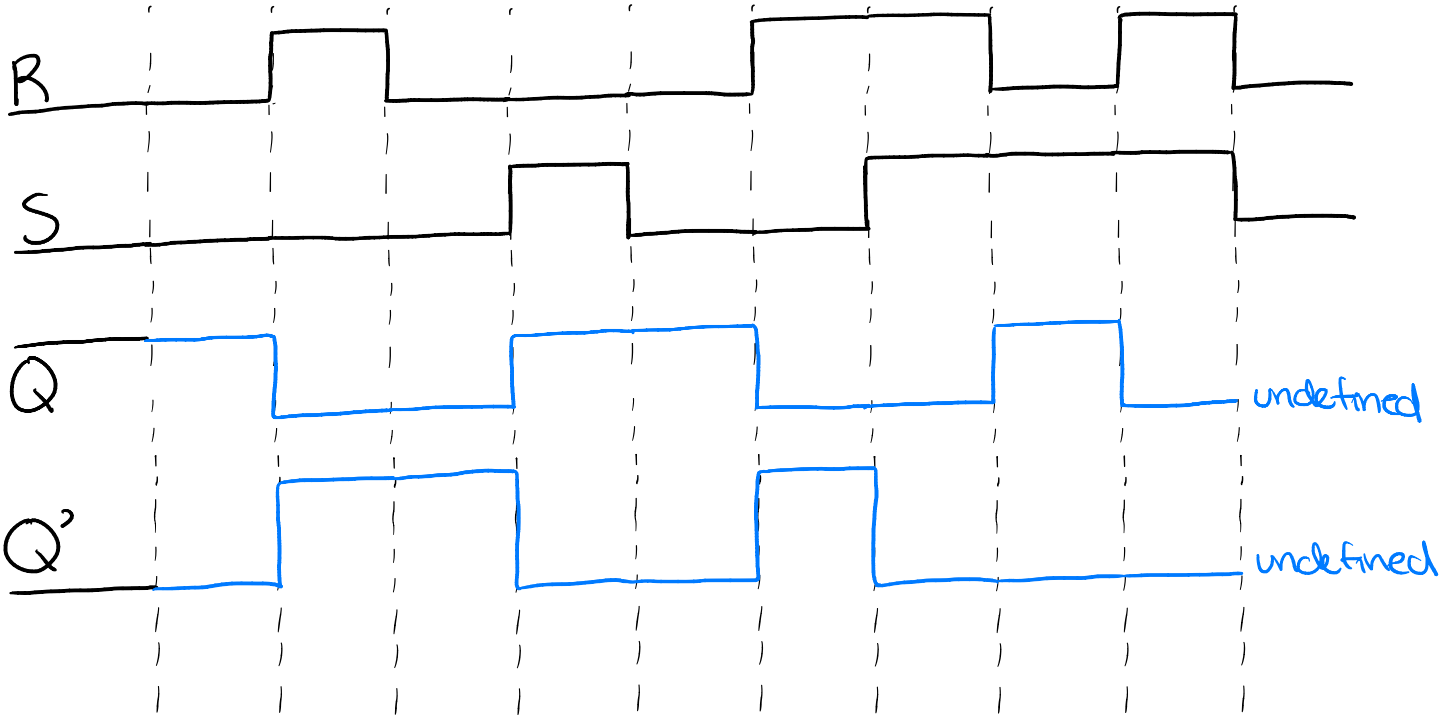
R	S	Q	Q next	Q next'
0	0	0	0	1
0	0	1	1	0
0	1	x	1	0
1	0	x	0	1
1	1	x	0	0

Ⓑ What does an SR latch do?
What does S stand for?
What does R stand for?

S stands for Set. R stands for Reset.

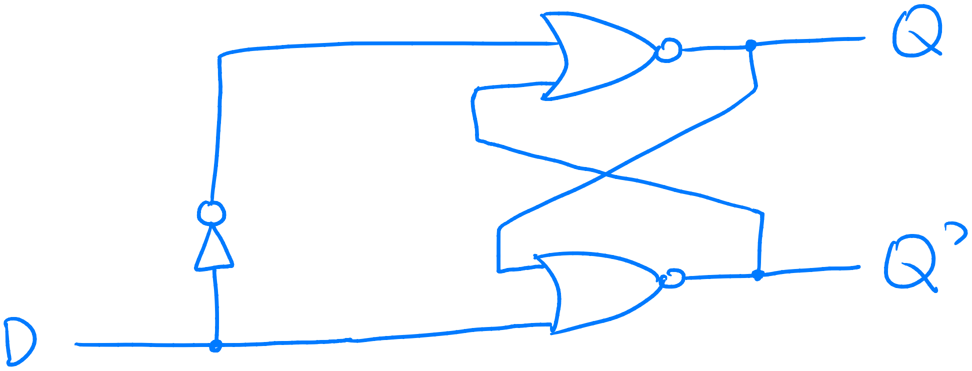
When S and R are not active, the SR latch holds onto its previous state.
When S is active, Q' = 0.
When R is active, Q = 0.

Ⓒ Complete the following timing diagram of an SR latch.
Assume the NOR gates have no delay.



③ D latch

Ⓐ Modify an SR latch by adding a new Input signal D. Wire D into S and D' into R.



Ⓑ What is the intended goal of the circuit you created in Ⓐ?

To avoid the "invalid" state of the SR latch where $Q = Q' = 0$.

Ⓒ Now, write the truth table for the circuit from Ⓐ.
Use D as the only input in your table.

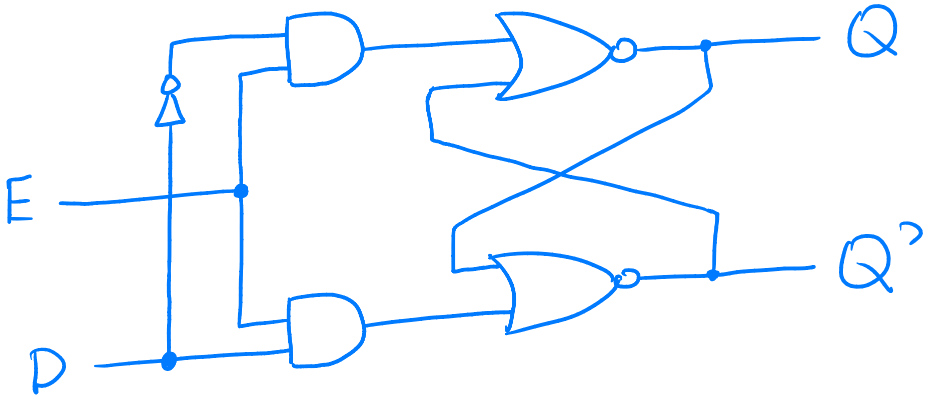
D	Q	Q_{next}	Q'_{next}
0	x	1	0
1	x	0	1

Ⓓ Why isn't the circuit from Ⓐ a latch?

There are no inputs that cause circuit to hold its previous state (like $S=R=0$ in the SR latch)

Ⓔ Modify the circuit from Ⓐ so that it "latches".

Hint: Use two AND gates and an enable signal called "E".



Ⓕ Write the truth table for the circuit from Ⓔ.

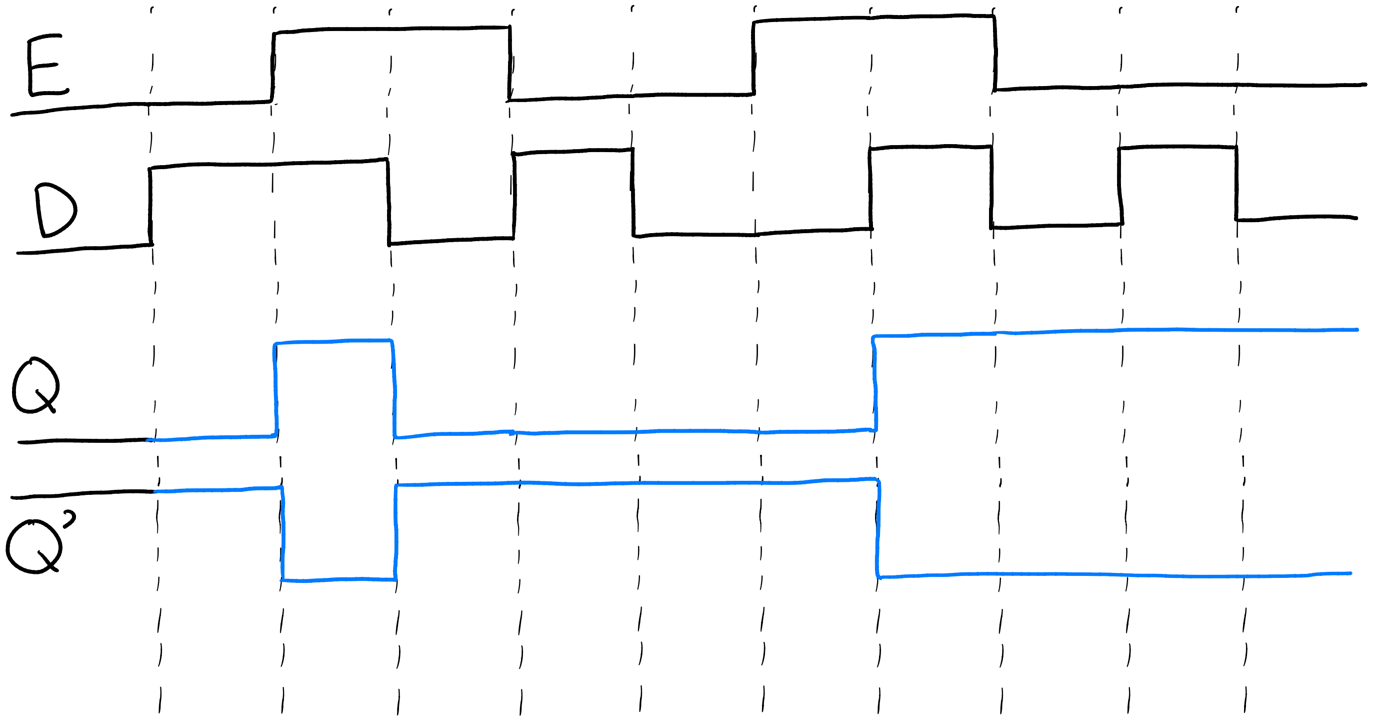
D	E	Q	Qnext	Q'next
0	X	0	0	1
0	X	1	1	0
1	0	X	0	1
1	1	X	1	0

Ⓖ What does a D latch do?
What does D stand for?

D stands for Data,

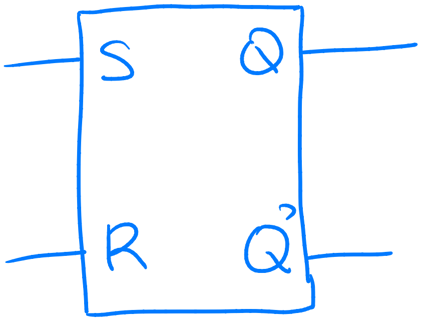
A D latch sets the next state to its input when enabled and hold its current state when disabled.

Ⓗ Complete the following timing diagram of an SR latch.
Assume the NOR gates have no delay.

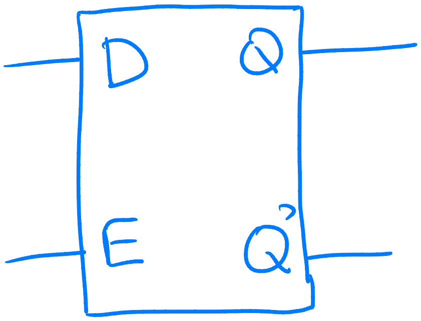


④ Black-box Latches

Ⓐ Draw the black-box version of the SR latch



Ⓑ Draw the black-box version of the D-latch (w/ enable)



Practice Exam 1: Boolean Logic and Combinational Circuits

Name:	
Student Number:	
Date:	

Exam Instructions

Write your name, your student number, and the date in the box above.

You may use a calculator, and scratch paper, and one front and back “cheat sheet”. If you submit your solutions (or intermediate work) on scratch paper, label each question clearly.

To receive full credit, you must **show all of your work.**

You will have **75 minutes** to complete this exam. The exam contains 6 problems (each with 2-3 parts).

Keep this exam booklet closed until the exam begins.

Problem 0: Numerical Representations

For each of the following hexadecimal numbers:

- Convert the hexadecimal number to 12-bit binary representation.
- Then, interpret the result as a signed binary number, and convert this number to its decimal representation.

a. $6AC_{16}$

12-bit Binary: 011010101100_2
Decimal: 1708_{10}

b. $9E7_{16}$

12-bit Binary: 100111100111_2
Decimal: -1561_{10}

c. 8_{16}

12-bit Binary: 000000000100_2
Decimal: 8_{10}

Problem 1: Boolean Algebra

Use Boolean Algebra to show that the following Boolean equation is true:

a. $(u \oplus v)(w \odot f) + u'w' = w'f'(u \oplus v) + v'w'f' + u'vf + v'f(u \odot w)$

Solution:

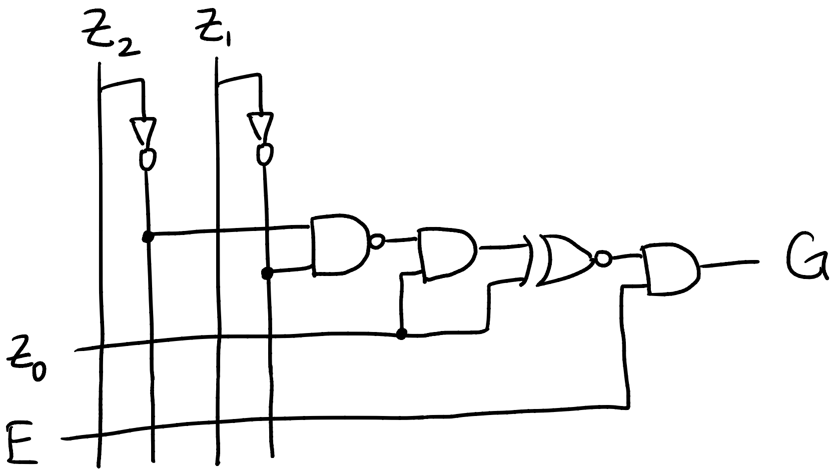
$$\begin{aligned}
LHS &= (u \oplus v)(w \odot f) + u'w' \\
&= (u'v + uv')(wf + w'f') + u'w'(v + v')(f + f') \\
&= u'vwf + u'vw'f' + uv'wf + uv'w'f' + u'vw'f + u'vw'f' + u'v'w'f + u'v'w'f' \\
&= u'vwf + u'vw'f' + uv'wf + uv'w'f' + u'vw'f + \cancel{u'vw'f'} + u'v'w'f + u'v'w'f' \\
&= u'vwf + u'vw'f' + uv'wf + uv'w'f' + u'vw'f + u'v'w'f + u'v'w'f' \\
&= u'vw'f' + uv'w'f' + u'v'w'f' + u'vwf + u'vw'f + uv'wf + u'v'w'f \\
&= u'vw'f' + uv'w'f' + uv'w'f' + u'v'w'f' + u'vwf + u'vw'f + uv'wf + u'v'w'f \\
&= (u'vw'f' + uv'w'f') + (uv'w'f' + u'v'w'f') + (u'vwf + u'vw'f) + (uv'wf + u'v'w'f) \\
&= w'f'(u'v + uv') + v'w'f'(u + u') + u'vf(w + w') + v'f(uw + u'w') \\
&= w'f'(u \oplus v) + v'w'f' + u'vf + v'f(u \odot w) = RHS
\end{aligned}$$

Problem 2: Analyzing Circuits

For each of the following circuit diagrams:

- Write the Boolean expression directly from the circuit. **Don't reduce this expression yet.** (use the algebraic symbols implied by the gates, e.g., use $\oplus$ for an XOR gate).
- Use Boolean algebra to reduce the Boolean expression to canonical sum-of-products form.
- Finally, interpret the inputs labeled Z as a single n-bit binary number
  (where n is the number of inputs labeled Z, e.g., if $Z = Z_1Z_0$, then n = 2).
  What decimal values of A will cause the output of the circuit to be 1?

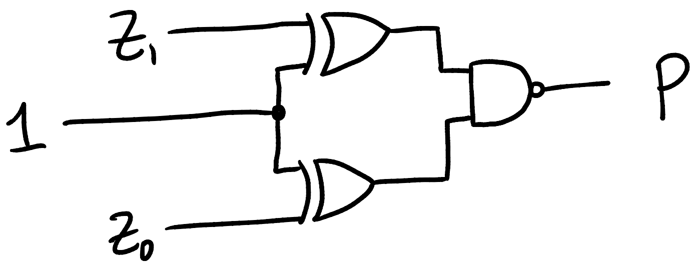
a.



$$\begin{aligned}
G &= E(Z_0 \odot Z_0(Z_2'Z_1')') \\
&= E(Z_0 \odot Z_0(Z_2 + Z_1)) \\
&= E(Z_0 \odot (Z_2Z_0 + Z_1Z_0)) \\
&= E(Z_0(Z_2Z_0 + Z_1Z_0) + Z_0'(Z_2Z_0 + Z_1Z_0)') \\
&= E(Z_2Z_0 + Z_1Z_0 + Z_0'(Z_2' + Z_0')(Z_1' + Z_0')) \\
&= E(Z_2Z_0 + Z_1Z_0 + Z_0') \\
&= EZ_2Z_0 + EZ_1Z_0 + EZ_0' \\
&= EZ_2Z_0(Z_1 + Z_1') + EZ_1Z_0(Z_2' + Z_2) + EZ_0'(Z_1 + Z_1')(Z_0 + Z_0') \\
&= EZ_2Z_1Z_0 + EZ_2Z_1'Z_0 + EZ_2'Z_1Z_0 + EZ_2Z_1Z_0 + EZ_2'Z_1'Z_0' + EZ_2'Z_1Z_0' + EZ_2Z_1'Z_0' + EZ_2Z_1Z_0' \\
&= EZ_2Z_1Z_0 + EZ_2Z_1'Z_0 + EZ_2'Z_1Z_0 + \cancel{EZ_2Z_1Z_0} + EZ_2'Z_1'Z_0' + EZ_2'Z_1Z_0' + EZ_2Z_1'Z_0' + EZ_2Z_1Z_0' \\
&= EZ_2Z_1Z_0 + EZ_2Z_1'Z_0 + EZ_2'Z_1Z_0 + EZ_2'Z_1'Z_0' + EZ_2'Z_1Z_0' + EZ_2Z_1'Z_0' + EZ_2Z_1Z_0' \\
&= EZ_2'Z_1'Z_0' + EZ_2'Z_1Z_0' + EZ_2'Z_1Z_0 + EZ_2Z_1'Z_0' + EZ_2Z_1'Z_0 + EZ_2Z_1Z_0' + EZ_2Z_1Z_0 \\
&= E(Z_2'Z_1'Z_0' + Z_2'Z_1Z_0' + Z_2'Z_1Z_0 + Z_2Z_1'Z_0' + Z_2Z_1'Z_0 + Z_2Z_1Z_0' + Z_2Z_1Z_0)
\end{aligned}$$

Assuming E is "enabled" ($E = 1$), then G will be 1 when $Z \in \{0, 2, 3, 4, 5, 6, 7\}$.

b.



$$
\begin{aligned}
P &= ((Z_1 \oplus 1)(Z_0 \oplus 1))' \\
&= (Z_1' Z_0')' \\
&= Z_1 + Z_0 \\
&= Z_1(Z_0 + Z_0') + (Z_1 + Z_1')Z_0 \\
&= Z_1Z_0 + Z_1Z_0' + Z_1Z_0 + Z_1'Z_0 \\
&= Z_1Z_0 + Z_1Z_0' + \cancel{Z_1Z_0} + Z_1'Z_0 \\
&= Z_1Z_0 + Z_1Z_0' + Z_1'Z_0 \\
&= Z_1'Z_0 + Z_1Z_0' + Z_1Z_0
\end{aligned}
$$

P will be 1 when $Z \in \{1, 2, 3\}$

Problem 3: Canonical Forms

Consider the following truth table for the function $\mathscr{W}$

a	b	c	d	$\mathscr{W}$	$\mathscr{W}'$
0	0	0	0	1	**0**
0	0	0	1	0	**1**
0	0	1	0	0	**1**
0	0	1	1	1	**0**
0	1	0	0	0	**1**
0	1	0	1	0	**1**
0	1	1	0	0	**1**
0	1	1	1	1	**0**
1	0	0	0	0	**1**
1	0	0	1	0	**1**
1	0	1	0	1	**0**
1	0	1	1	0	**1**
1	1	0	0	0	**1**
1	1	0	1	0	**1**
1	1	1	0	1	**0**
1	1	1	1	0	**1**

a. Fill out the column for $\mathscr{W}'$ in the table above

b. Write the Boolean expression for $\mathscr{W}$ in SOP form

$$\mathscr{W} = a'b'c'd' + a'b'cd + a'bcd + ab'cd' + abcd'$$

c. Write the Boolean expression for $\mathscr{W}'$ in POS form

$$\mathscr{W}' = (a+b+c+d)(a+b+c'+d')(a+b'+c'+d')(a'+b+c'+d)(a'+b'+c'+d)$$

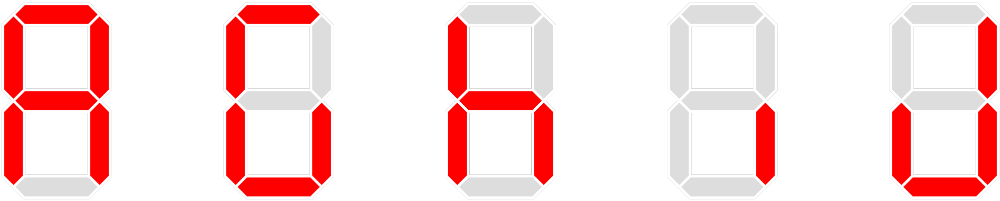
Problem 4: 7-Segment Display

For this problem, you will use a 7-segment display to display the letters **A**, **G**, **h**, **i**, and **J**.

Your circuit will:

- input a 3-bit **signed** binary number X
- output the 7 signals for the 7-segment display (a,b,c,d,e,f,g) based on the following logic:
  - if $X = -4$, the display output should be **A**
  - if $X = -2$, the display output should be **G**
  - if $X = 0$, the display output should be **h**
  - if $X = 1$, the display output should be **i**
  - if $X = 3$, the display output should be **J**

The letters **A**, **G**, **h**, **i**, and **J** should look like this:



a. Draw the truth table for this circuit.

Note: Your input signals should represent the 3-bit binary number X, and your output signals should represent the input signals to the 7-segment display.

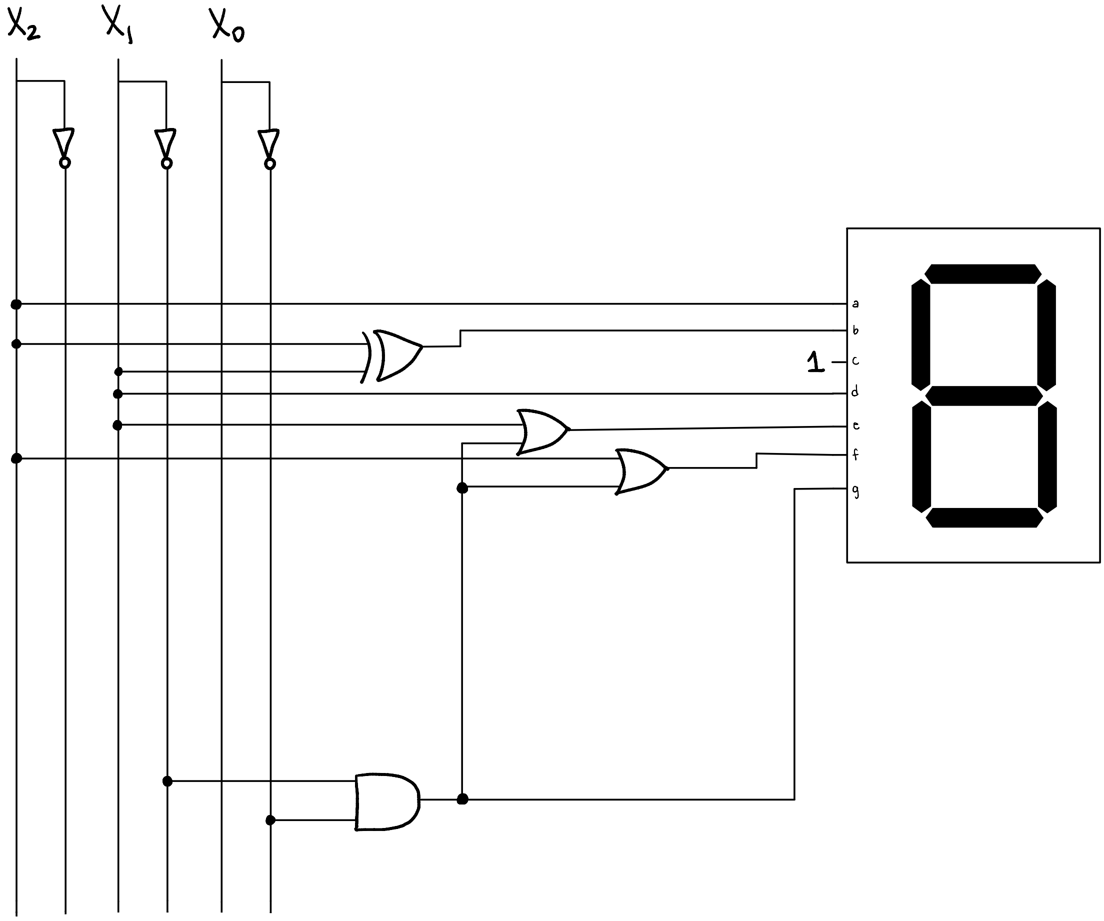
Decimal Value of X	X_2	X_1	X_0	a	b	c	d	e	f	g
0	0	0	0	0	0	1	0	1	1	1
1	0	0	1	0	0	1	0	0	0	0
2	0	1	0	×	×	×	×	×	×	×
3	0	1	1	0	1	1	1	1	0	0
-4	1	0	0	1	1	1	0	1	1	1
-3	1	0	1	×	×	×	×	×	×	×
-2	1	1	0	1	0	1	1	1	1	0
-1	1	1	1	×	×	×	×	×	×	×

b. Write the Boolean expressions for each of the 7-segment display input signals

$$a = X_2$$
$$b = X_2 \oplus X_1$$
$$c = 1$$
$$d = X_1$$
$$e = X_1'X_0' + X_1$$
$$f = X_1'X_0' + X_2$$
$$g = X_1'X_0'$$

c. Draw the complete circuit that implements the expressions you derived in part b

Note: Connect your outputs to the inputs of the 7-segment display provided.



Problem 5: Using Small Boxes to Build Big Boxes

For this problem, you will **use a "black-box" circuit to create a larger combinational circuit**.

The "black-box" circuit has:

- two input signals: A and B, and
- two output signals: S and G with the following logic:
  - if $A > B$, then $S = 0$ and $G = 1$
  - if $A < B$, then $S = 1$ and $G = 0$
  - if $A = B$, then $S = 0$ and $G = 0$

Your goal is to create a circuit that has:

- four input signals: A, B, C, and D, and
- one output signal: F with the following logic:
  - if $A > B$, then $F = C$
  - if $A < B$, then $F = D$
  - if $A = B$, then $F = C + D$

Your circuit will pass the A and B signals to the appropriate inputs of the "black-box" circuit.
Then, you will use the resulting output signals (S and G), in conjunction with the remaining input signals (C and D) to generate the output signal for the larger combinational circuit (F).

Continue to the next page...

a. Fill out the truth table below

Note: The input columns on this table are C, D, S, and G.

C	D	S	G	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	×
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	×
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	×
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	×

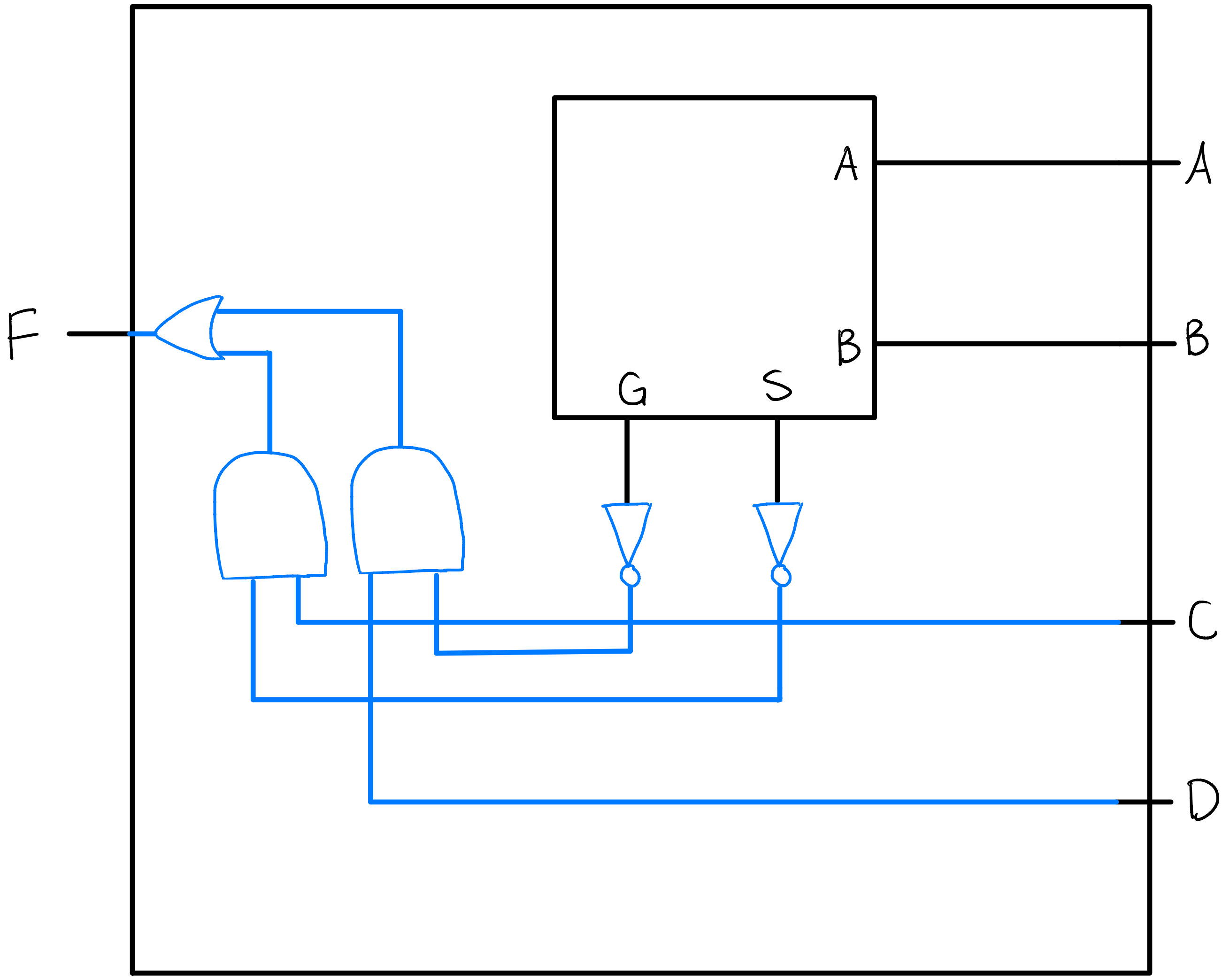
b. Use a K-map to write the Boolean expression for the output signal

You may want to draw your K-map to the right of the truth table above, and just write the expression here. **Make sure the subcubes you draw are clear.**

$$F = DG' + CS'$$

c. Use lines and basic logic gates to complete the circuit below.

Note: Your circuitry should fit inside the box for the larger combinational circuit.



Practice Exam 2: MUX, ALUs, Sequential Circuits and FSMs

Name:	
Student Number:	
Date:	

Exam Instructions

Write your name, your student number, and the date in the box above.

You may use a calculator, and scratch paper, and one front and back "cheat sheet". If you submit your solutions (or intermediate work) on scratch paper, label each question clearly.

To receive full credit, you must **show all of your work.**

You will have **120 minutes** to complete this exam. The exam contains 5 problems.
Keep this exam booklet closed until the exam begins.

Problem 0: Binary Arithmetic and Overflow [5 points each]

Perform each of the following 6-bit binary arithmetic operations, then:

- Interpret the operands and result as **unsigned** decimal numbers and state whether overflow occurred.
- Interpret the operands and result as **signed** decimal numbers and state whether overflow occurred.

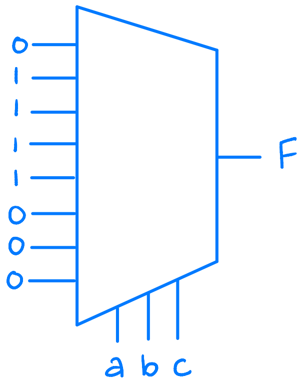
Operation	**Unsigned Overflow**	**Signed Overflow**
110001 + 101101 = 011110	49 + 45 = 30; Overflow	-15 + -19 = 30; Overflow
011011 + 010011 = 101110	27 + 19 = 46; No Overflow	27 + 19 = -18; Overflow
011001 + 101010 = 000011	25 + 42 = 3; Overflow	25 + (-22) = 3; No Overflow

Problem 1: Multiplexer Circuits [20 points]

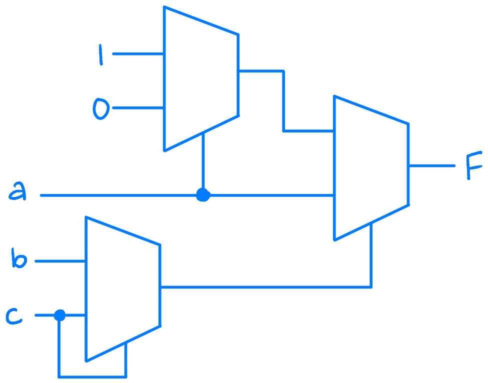
$$F(a,b,c) = a' \odot (b+c)$$

Draw a circuit for the Boolean equation above:

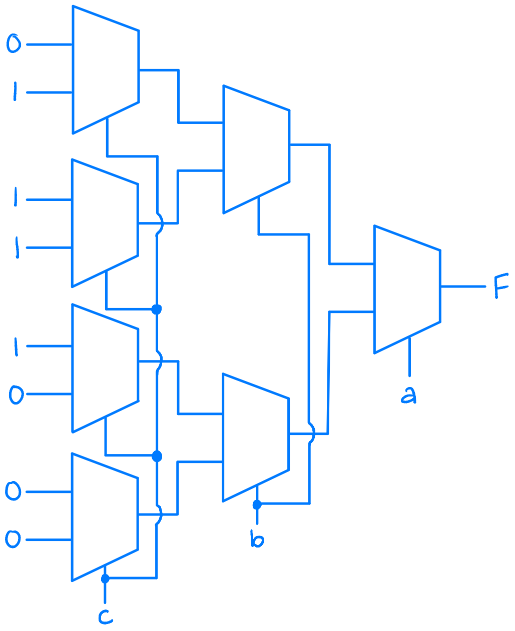
(a) using an 8-to-1 multiplexer [10 points]


0
1
1
1
1
0
0
0
F
a b c

(b) using an 8-to-1 multiplexer [10 points]



OR


0
1
1
1
1
0
0
0
F
a
b
c

Problem 2: Multiplexer Expressions [10 points]

Choose **ONE** of the following circuits containing multiplexers:

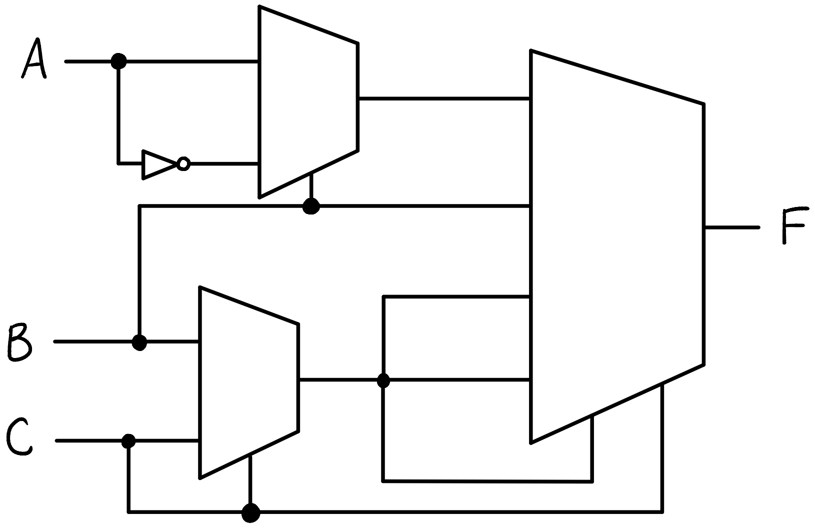
(a) Write a **reduced** Boolean expression for **F**

(b) Draw a new circuit implementing your expression for F using only the following gate types:

- NOT
- 2-input AND
- 2-input OR
- 2-input XOR

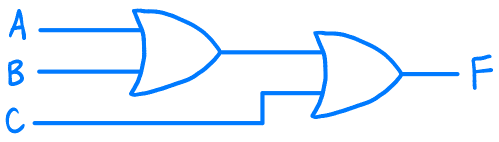
For full credit, use the fewest number of gates in your circuit as possible.

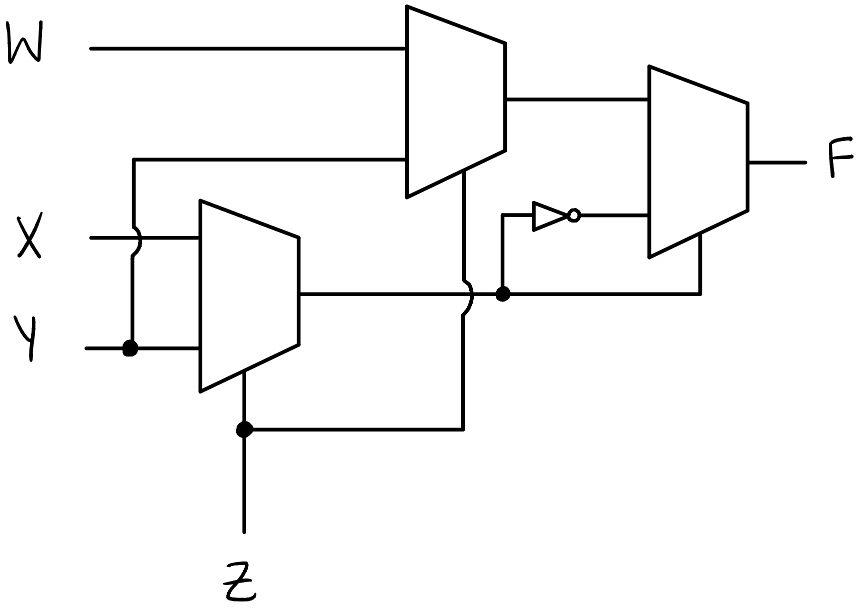
For +10 points extra credit, do the above for BOTH circuits



From the circuit above (with a couple shortcuts), then using Boolean algebra:

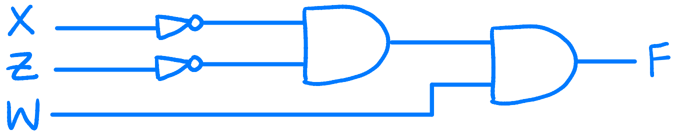
$$\begin{aligned} F &= B + C + C'(A \oplus B) + CB \\ &= B + C + C'(A \oplus B) \\ &= B + C + A \oplus B \\ &= B + C + AB' + A'B \\ &= B + C + AB' \\ &= B + C + A \end{aligned}$$





From the circuit above (with NO shortcuts), then using Boolean algebra:

$$\begin{aligned} F &= (Z'X + ZY)'(Z'W + ZY) + (Z'X + ZY)'(Z'X + ZY) \\ &= (Z'X + ZY)'(Z'W + ZY) \\ &= (Z + X')(Z' + Y')(Z'W + ZY) \\ &= (Z + X')(Z'W + Z'WY) \\ &= (Z + X')Z'W \\ &= X'Z'W \end{aligned}$$



Problem 3: Arithmetic Logic Units [30 points]

a. Draw the LE, AE, and CE circuits for an ALU that has the following operations. reduce all of the equations (e.g., using K-maps). [15 points]

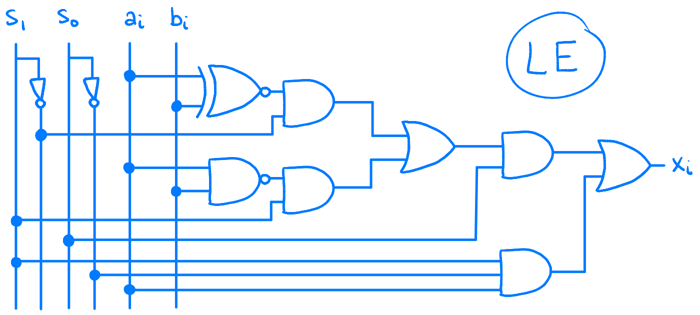
s_1	s_0	**Operations**	x_i	y_i	c_0
0	0	$-B$	0	b_i'	1
0	1	A XNOR B	$a_i \odot b_i$	0	0
1	0	$A - B$	a_i	b_i'	1
1	1	A NAND B	$a_i' + b_i'$	0	0

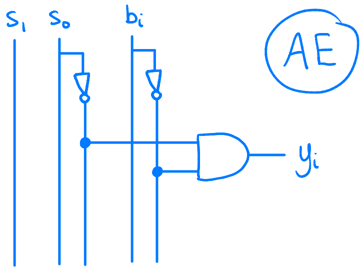
Using sum-of-products and Boolean Algebra (you could use k-maps):

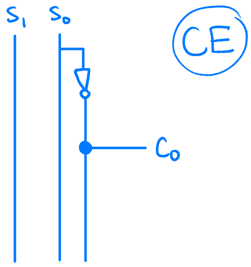
$$\begin{aligned} x_i &= s_1' s_0 (a_i \odot b_i) + s_1 s_0' a_i + s_1 s_0 (a_i' + b_i') \\ &= s_0 (s_1' (a_i \odot b_i) + s_1 (a_i' + b_i')) + s_1 s_0' a_i \end{aligned}$$

$$\begin{aligned} y_i &= s_1' s_0' b_i' + s_1 s_0' b_i' \\ &= s_0' b_i' (s_1' + s_1) \\ &= s_0' b_i' \end{aligned}$$

$$\begin{aligned} c_0 &= s_1' s_0' + s_1 s_0' \\ &= s_0' (s_1' + s_1) \\ &= s_0' \end{aligned}$$





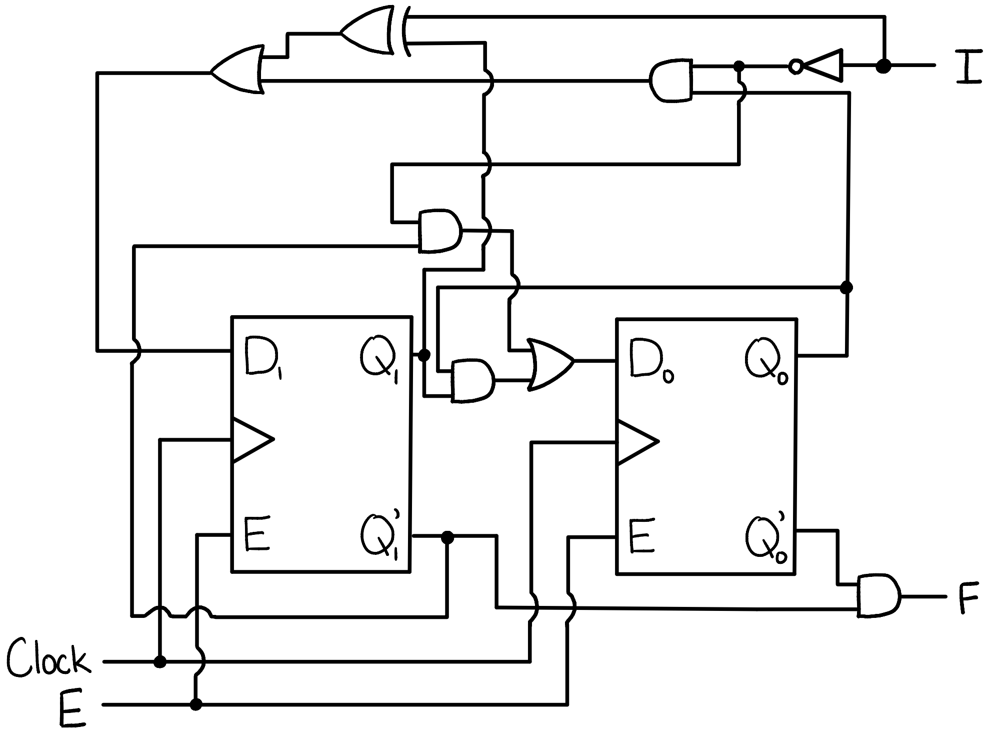


b. Determine the four functions (in the correct order) that are implemented by an ALU with the following equations for its LE, AE, and CE. [15 points]

$$\begin{aligned}x_i =& s_1(s_0 \odot a_i) + (s_1 \oplus s_0)b_i' \\ y_i =& s_1 s_0 \\ c_0 =& s_1'\end{aligned}$$

s_1	s_0	x_i	y_i	c_0	**Operations**
0	0	0	0	1	1
0	1	b_i'	0	1	$-B$
1	0	$a_i' + b_i'$	0	0	A NAND B
1	1	a_i	1	0	$A - 1$

Problem 4: Analyzing Finite State Machines [25 points]



Use the sequential circuit above to complete the questions below:

a. Write the next state equation(s) for the circuit above.

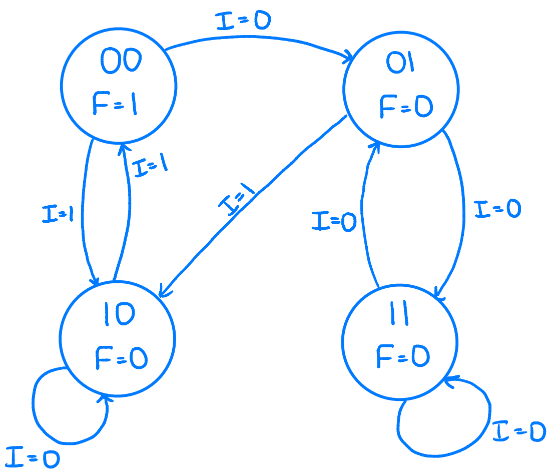
$$Q_{1next} = I'Q_0 + I \oplus Q_1$$
$$Q_{0next} = Q_1Q_0 + I'Q_1'$$

b. Write the next state table using the equations from part a.

	$Q_{1next}Q_{0next}$	
Q_1Q_0	$I = 0$	$I = 1$
00	01	10
01	11	10
10	10	00
11	11	01

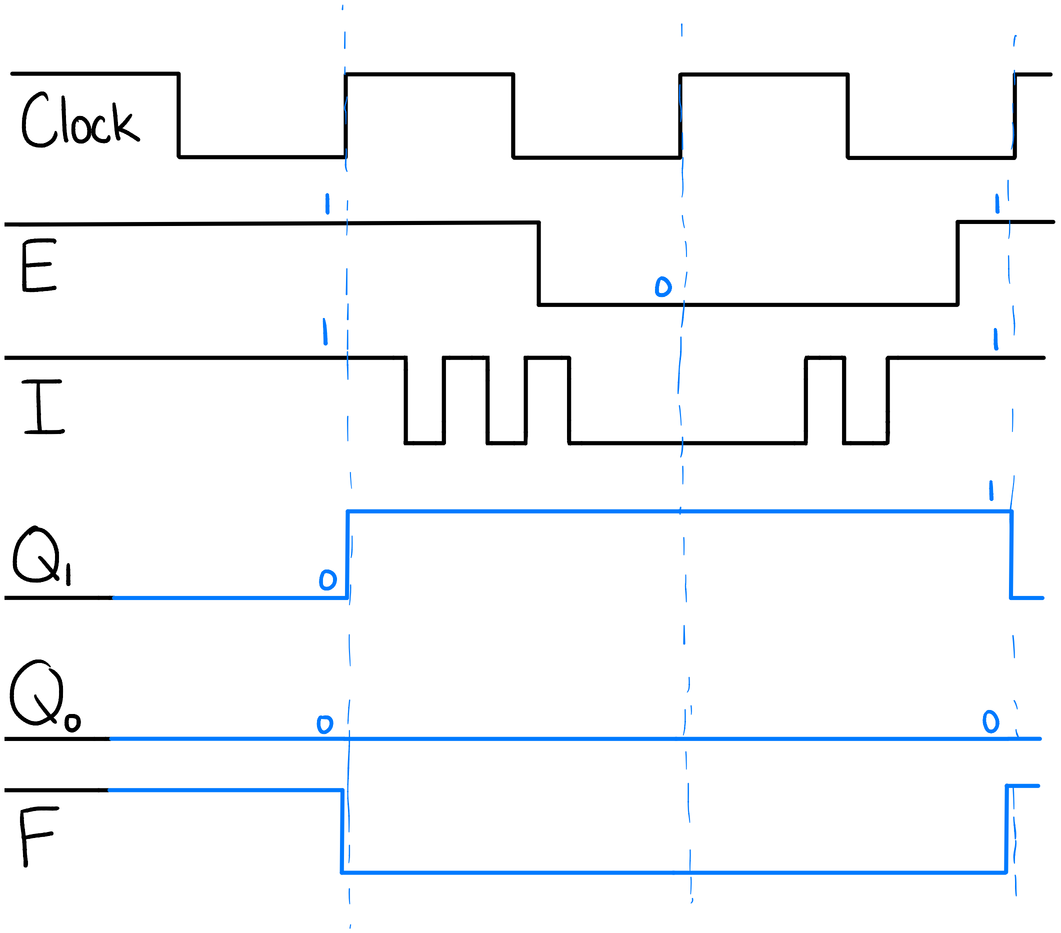
c. Draw the (Moore) state diagram for the circuit.

Make sure to include the necessary states, inputs, and outputs in your diagram.



I=0
00
F=1
01
F=0
I=1
I=1
I=1
I=0
I=0
10
F=0
11
F=0
I=0
I=0

d. Complete the following timing diagram based on the circuit.



Clock
E
I
Q1
Q0
F
1
1
0
1
1
0
1
0
0
0